Delta University Scientific Journal
Journal home page: https://dusj.journals.ekb.eg

Assigning accuracy for Cars Detection from High-Resolution Satellite Images Using Different Machine Learning Models

Mariem A. elhalawani[1], Mahmoud El-Mewafi[2], Fawzi Zarzora[3] and Mohammed Shaaban[3]

[1]Teaching Assistant, Basic Science department, Delta University for Science and Technology, 7731168 Gamasa, Egypt.
[2]Professor, Civil Engineering department, Mansoura university,35516 Mansoura, Egypt.
[3,4]Assistant Professor, Civil Engineering department, Mansoura university,35516 Mansoura, Egypt.
[4]Assistant Professor, Civil Engineering department, Delta University for Science and Technology, 7731168Gamasa, Egypt.
**Corresponding author. E-mail: mariem_elhalawani@yahoo.com*

ABSTRACT
High-resolution satellite imagery provides a wealth of detailed visual information that can be leveraged for various machine learning applications. In this study, we present a deep learning-based approach for car classification using high-resolution satellite images. Utilizing the powerful capabilities of Tensor Flow layers, we design and implement a convolutional neural network (CNN) to accurately identify and classify different types of cars from satellite imagery. The process involves the collection of a diverse dataset of satellite images containing vehicles, followed by rigorous data pre-processing and augmentation to enhance model robustness. The CNN architecture is optimized through hyper parameter tuning and trained on a labeled dataset, achieving high accuracy in classifying vehicles into predefined categories such as sedans, SUVs, and trucks. Our results demonstrate the effectiveness of using deep learning models with TensorFlow layers for car classification tasks, highlighting the potential for broader applications in urban planning, traffic management, and automated vehicle detection from satellite imagery.

Keywords: *High-Resolution Satellite Images - Deep Learning - Car Classification - TensorFlow*

1. Introduction

In the rapidly evolving automotive industry, the ability to automatically classify cars based on various attributes has become increasingly important. Car classification involves categorizing vehicles into predefined classes, such as sedans, SUVs, trucks, and convertibles, based on their features as shown in figure (1). This capability can be leveraged for a range of applications, from enhancing autonomous driving systems to streamlining vehicle inventory management for dealerships.

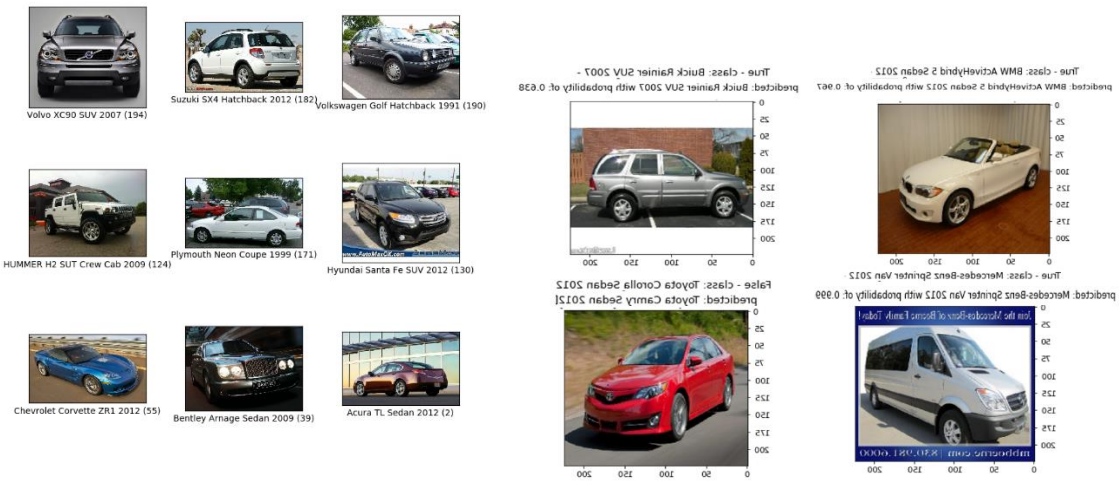


Figure (1): car classification involves categorizing vehicles into predefined classes.

The goal of this project is to develop a car classification system using machine learning techniques. By training a model on a dataset of labelled car from satellite images, we aim to create an accurate and efficient classifier that can predict the class of a car from an input image. This system can be implemented using popular machine learning frameworks such as Tensor Flow or PyTorch, and will involve several key steps, including:

1. Data Collection: Gathering a diverse and representative dataset of car images, labelled with their respective two classes (with car and without car).
2. Data Pre-processing: Cleaning and preparing the data for training by resizing images, normalizing pixel values, and augmenting the dataset to improve model robustness.
3. Model Selection: Choosing an appropriate machine learning model architecture, such as a convolutional neural network (CNN), which is well-suited for image classification tasks.
4. Model Training: Training the model on the pre-processed dataset, tuning hyper parameters, and validating the model's performance using a validation set.
5. Model Evaluation: Assessing the model's accuracy, precision, recall, and other relevant metrics on a test set to ensure its generalizability to new, unseen data.
6. Deployment: Integrating the trained model into a practical application, such as a web or mobile app, where users can upload car images and receive classification results in real-time.

By the end of this project, we will have a functional car classification system capable of accurately identifying the class of a car from an image. This technology has the potential to enhance various automotive industry processes, improve user experiences, and contribute to the development of more advanced vehicle-related technologies.

2. Data Collection for Cars Classification

Data collection is a critical step in the development of a robust car classification system. The quality, diversity, and volume of the dataset directly influence the performance and accuracy of the machine learning model. For car classification, the dataset primarily consists of labelled images of cars belonging to various classes such as sedans, SUVs, trucks, convertibles, and more. Here's a detailed overview of the data collection process to build deep learning model for car classification:

1. Identify Data Sources

In recent years, several remote sensing datasets have been developed. This paper focuses on the extraction of small or tiny objects (cars) from the satellite image with high resolution (30 cm from SAS planet software as shown in figure (2), which consists of four categories roads, buildings, trees and cars).

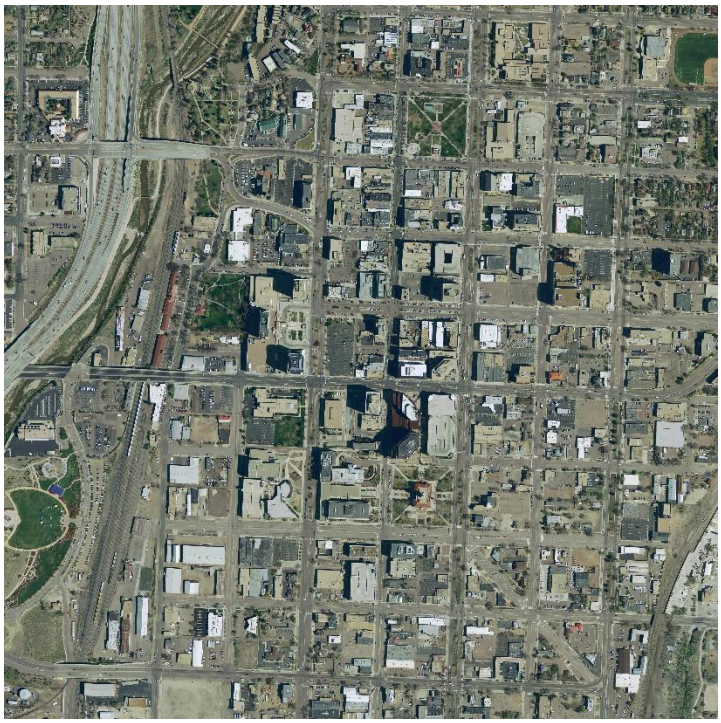


Figure (2): high resolution satellite image in united states

2.1. Ensure Data Diversity

To train a model that generalizes well, the dataset should include a wide range of images covering:

- Different Classes: Ensure all car classes of interest are well-represented.
- Various Angles: Include images captured from different angles (front, side, rear) and perspectives (close-up, distant).
- Multiple Conditions: Collect images taken under diverse lighting conditions, weather conditions, and backgrounds.
- Different Makes and Models: Include images of cars from various manufacturers and models within each class.

2.2. Labelling and Annotation

Accurate labelling is crucial for supervised learning:

- Manual Annotation: Manually label the images with the correct car class. This can be done using annotation tools like Labelling.
- Automated Annotation: Use pre-trained models to assist in annotating images, followed by manual verification to ensure accuracy.
- Crowdsourced Annotation: Employ crowdsourcing platforms to label images, ensuring to implement quality control measures such as redundancy (having multiple annotators label the same image) and consensus algorithms.

In this study. However, as shown in figure (3) due to the large image size in dataset, direct training is not feasible. Therefore, we have cropped the image to a size of 125 X 125, resulting in a total of 340 images. These images were subsequently divided into three sets according to the train: Val: test ratio, with a split of 1:1:1.

Figure (3) : images in size 125 * 125

2.3. Data Cleaning and Pre-processing

Before using the images for model training, it's essential to pre-process the data:

- Removing Duplicates: Ensure there are no duplicate images in the dataset.
- Handling Imbalance: If certain classes are underrepresented, consider techniques like data augmentation (flipping, rotating, scaling images) or collecting additional images to balance the dataset.
- Standardization: Normalize the images to a standard size and resolution to ensure consistency across the dataset.
- Annotation Quality Check: Verify the accuracy of labels and correct any miss annotations.

2.4. Deep learning program

Load data

Loading data efficiently is a crucial step in training a deep learning model. The process involves several stages to ensure that the data is ready for the model to learn from. Initially, data needs to be read from storage, which could be local files, cloud storage, or a database. This raw data often consists of images, text, or other formats, depending on the task at hand. For image-based tasks like car classification, images are typically loaded using libraries such as OpenCV, PIL, or specialized data loading utilities provided by deep learning frameworks like TensorFlow and PyTorch. Once loaded, the data undergoes pre-processing steps, which may include resizing, normalization, augmentation, and conversion to the appropriate format for model ingestion. In TensorFlow, for example, the tf.data API provides tools to build efficient data pipelines, supporting operations like batching, shuffling, and parallel processing (Shorten, C., & Khoshgoftaar, T. M, 2019).

Pre-process data

Ensuring in our study that input images are properly prepared during the loading stage is crucial for the success of a deep learning classification model. This process begins with resizing the images to a consistent size that matches the input dimensions expected by the model, such as 255x255 pixels for many convolutional neural networks (CNNs) (TensorFlow, 2024). Normalizing pixel values, typically by scaling them to a range of 0 to 1, helps stabilize and accelerate the training process by ensuring that the inputs have a standard distribution.in this study 0 for with car class and 1 to without car class as shown in figure (4). Additionally, data augmentation techniques such as random rotations, flips, and crops can be applied to increase the diversity of the training data and improve the model's robustness to variations in the input. These steps are often implemented using specialized functions within deep learning frameworks like TensorFlow's tf.image module or PyTorch's transforms module (Chollet, F, 2018). Proper pre-processing not only enhances the model's ability to learn meaningful patterns but also helps in reducing

overfitting by exposing the model to a wider array of image variations during training. By ensuring that images are consistently formatted and augmented, we lay a solid foundation for building accurate and reliable classification models.

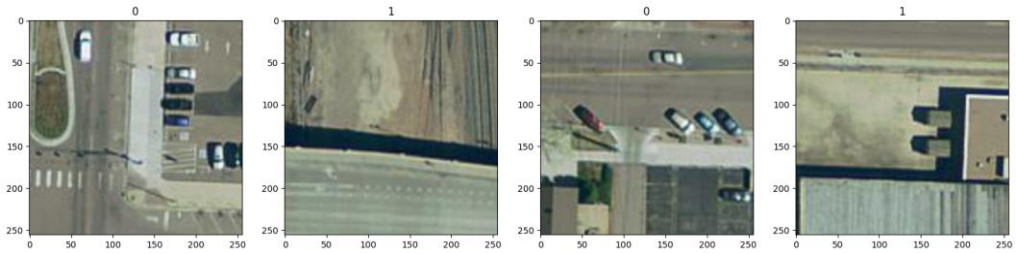


Figure (4): resizing and scaling the image

2.5. Data Splitting

Divide the collected dataset into three subsets:

- Training Set: The largest subset used to train the model.
- Validation Set: Used to tune hyperparameters and validate the model during training.
- Test Set: A separate subset used to evaluate the final model's performance on unseen data.

Split data

Effective data splitting is crucial for developing a reliable car classification model in deep learning. The dataset is typically divided into three main subsets: training, validation, and test sets (Géron, A, 2019). The training set, comprising about 70% of the total data, is used to train the model. This subset is where the model learns to identify features and patterns associated with different car classes. The validation set, around 20% of the data, is used during the training process to tune hyper parameters and prevent overfitting by providing an unbiased evaluation of the model's performance. Finally, the test set, which also constitutes about 10% of the data as shown in figure (5), is used to assess the model's generalizability to new, unseen data after the training is complete. This ensures that the model performs well not only on the data it was trained on but also on any new data it encounters. In practice, the data should be split randomly but in a stratified manner to ensure that each subset has a proportional representation of all car classes. This stratification is essential to maintain the distribution of car classes across the training, validation, and test sets, thereby ensuring that the model learns from a representative sample of the data and performs consistently across all subsets (Goodfellow, I., 2016).

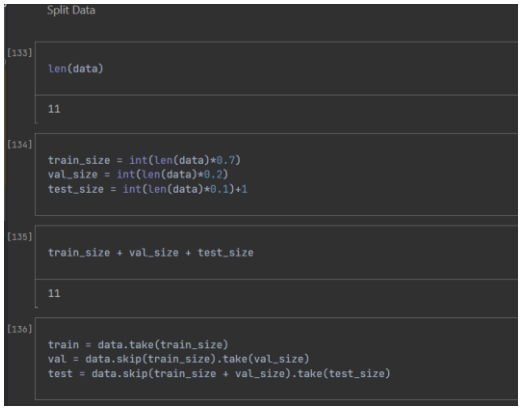


Figure (5): split data (training set 70%- validation set 20% -test set 10%)

3. Build deep learning model

Building a deep learning model for car classification involves designing a neural network architecture using TensorFlow and its high-level Keras API as shown in figure (6). This process in this study begins with defining the model's architecture, typically starting with an input layer that matches the shape of the pre-processed car images. The next step involves adding convolutional layers (Conv2D), which are essential for extracting features from

images through learnable filters. These layers are often followed by activation functions like ReLU (Rectified Linear Unit) to introduce non-linearity. Pooling layers (MaxPooling2D) are then used to reduce the spatial dimensions, making the computation more efficient and reducing overfitting (Srivastava, N., 2014). To further enhance the model's ability to learn complex patterns, several convolutional and pooling layers can be stacked. After feature extraction, the data is flattened and fed into fully connected (dense) layers, which perform the final classification. The output layer typically uses a softmax activation function to provide probabilities for each car class. Regularization techniques such as dropout layers can be included to prevent overfitting. Once the architecture is defined, the model is compiled with an appropriate loss function (e.g., categorical crossentropy for multi-class classification) and an optimizer like Adam. The model is then trained on the labeled car dataset using the fit method, and its performance is evaluated using validation data to fine-tune the parameters and improve accuracy. This structured approach using TensorFlow Keras layers allows for the creation of a robust and efficient car classification model.

Layer (type)	Output Shape	Param #
conv2d_9 (Conv2D)	(None, 254, 254, 16)	448
max_pooling2d_6 (MaxPooling2D)	(None, 127, 127, 16)	0
conv2d_10 (Conv2D)	(None, 125, 125, 32)	4,640
max_pooling2d_7 (MaxPooling2D)	(None, 62, 62, 32)	0
conv2d_11 (Conv2D)	(None, 60, 60, 16)	4,624
max_pooling2d_8 (MaxPooling2D)	(None, 30, 30, 16)	0
flatten_2 (Flatten)	(None, 14400)	0
dense_4 (Dense)	(None, 256)	3,686,656
dense_5 (Dense)	(None, 1)	257

Total params: 3,696,625 (14.10 MB)

Trainable params: 3,696,625 (14.10 MB)

Non-trainable params: 0 (0.00 B)

Figure (6): TensorFlow and its high-level Keras API for building car classification model

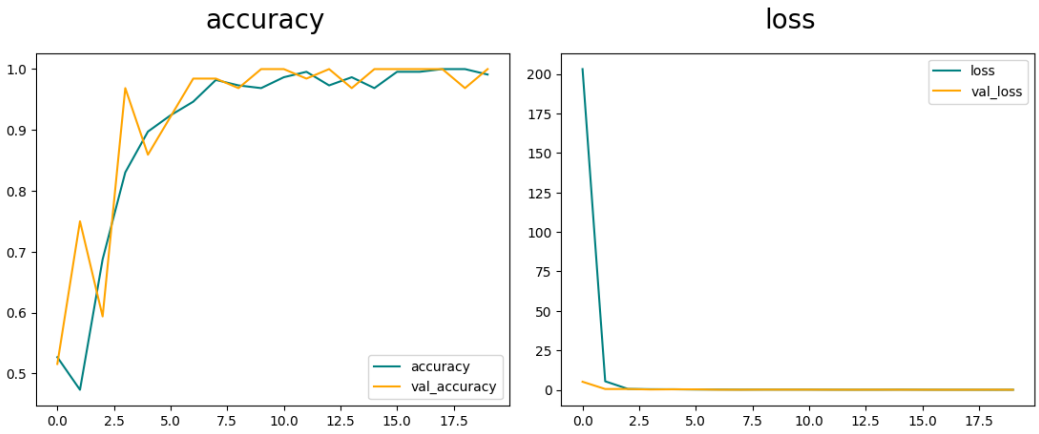
4. Train

Once a deep learning model for car classification is built, the next step is to train the model using the prepared dataset. Training involves feeding the pre-processed data into the model in batches, allowing the model to learn the distinguishing features of different car classes through a process called backpropagation (Rumelhart, D. E., 1986). The model's weights are iteratively adjusted to minimize the loss function, which quantifies the difference between the predicted and actual classes. This process typically involves several epochs, where each epoch represents a complete pass through the entire training dataset. During training, it is crucial to use techniques such as learning rate scheduling, early stopping, and regularization to enhance model performance and prevent overfitting. Learning rate scheduling adjusts the learning rate during training to ensure efficient convergence, while early stopping halts training when the model's performance on a validation set stops improving, thus preventing overfitting. Regularization methods, such as dropout, add robustness to the model by randomly deactivating neurons during training. Tools like TensorFlow's Model.fit or PyTorch's training loop handle these processes, enabling efficient and effective training of deep learning models for car classification (Bengio, Y, 2012).

Evaluate performance

After training a deep learning model for car classification, performance evaluation is essential to determine its effectiveness and generalization capability. This evaluation involves several metrics and techniques to assess how well the model performs on unseen data. Commonly used metrics include accuracy, precision, recall, and the F1-

score (Sokolova, M., & Lapalme, G, 2009). Accuracy measures the overall correctness of the model, while precision and recall provide insights into the model's performance on specific classes, highlighting its ability to avoid false positives and false negatives, respectively. The F1-score, a harmonic mean of precision and recall, offers a balanced measure when dealing with imbalanced datasets. Additionally, a confusion matrix is used to visualize the model's predictions compared to the actual labels, revealing patterns of misclassification.



(A) (B)

Figure (7): A) the accuracy of deep learning model, B) the loss resulted from train and validation set

Test

it's important to conduct robustness tests, such as assessing the model under different lighting conditions, occlusions, and image resolutions, to ensure it performs well across diverse scenarios. The results of these evaluations guide further refinements and adjustments to the model, such as tweaking hyper parameters, enhancing data augmentation techniques, or modifying the network architecture. Through this thorough testing and performance evaluation, the model can be optimized to achieve high accuracy and robustness in real-world car classification applications.

```
6]
yhat = model.predict(np.expand_dims(resize/255, 0))
print(yhat)
if yhat >= 0.5:
    print('----------------------------')
    print('The Image Test shows NO CARS')
    print('----------------------------')
else:
    print('--------------------------')
    print('The Image Test shows CARS')
    print('--------------------------')
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
[[0.5689186]]
----------------------------
The Image Test shows NO CARS
----------------------------
```

Figure (8): evaluation and adjustment of the model

Therefore, any image is recalled and placed in the appropriate scale for the model. The model is asked to test the image and classify it as whether it contains a car or not. If it recognizes the absence of a car by a percentage greater than 50%, then the model is able to classify the image without a car.

Conclusion

In conclusion, utilizing TensorFlow's high-level tf.keras. layers API for car classification has proven to be an effective approach. By leveraging pre-built layers and model architectures, such as convolutional neural networks (CNNs), we can construct deep learning models that are both powerful and flexible. The modular nature of TensorFlow's layers allows for easy customization and fine-tuning of models to achieve optimal performance on the car classification task. Through a well-defined workflow involving data collection, preprocessing, and model training, we can develop robust classifiers capable of accurately categorizing various car types from images.

References

Bengio, Y. Practical recommendations for gradient-based training of deep architectures. arXiv preprint arXiv:1206.5533, 2012.

Chollet, F.. Deep Learning with Python. Manning Publications. Deep Learning with Python, 2018.

Géron, A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media, 2019.

Goodfellow, I., Bengio, Y., & Courville, A. Deep Learning. MIT Press, 2016.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. Learning representations by back-propagating errors. Nature, 1986. 323(6088), 533-536

Shorten, C., & Khoshgoftaar, T. M. A survey on Image Data Augmentation for Deep Learning. Journal of Big Data. 2019, 6(1), 1-48.

Sokolova, M., & Lapalme, G. A systematic analysis of performance measures for classification tasks. Information Processing & Management, 2009. 45(4), 427-437.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 2014. 15(1), 1929-1958

TensorFlow. tf.image: Image processing. TensorFlow Documentation. 2024.