MOBILE APPLICATION DEVELOPMENT
PROJECT REPORT

SAI VISHWAS PATHEM
A20390102

Introduction to Core ML

It allows developers to seamlessly integrate trained machine learning models into their apps to enable intelligent features such as image recognition, natural language processing, and predictive analysis.

Development of Core ML

Apple introduced Core ML in 2017 as part of its efforts to bring machine learning capabilities to its ecosystem of devices. Core ML was designed to provide a unified and efficient way for developers to leverage the power of machine learning models without needing extensive expertise in machine learning algorithms or techniques.

Working of Core ML

Core ML works by allowing developers to convert trained machine learning models from popular frameworks such as TensorFlow or PyTorch into a format that can be efficiently run on Apple devices. This format, known as Core ML model (.mlmodel), is optimized for performance on Apple's hardware, including the CPU, GPU, and Neural Engine.

Once a Core ML model is integrated into an app, the app can use Core ML APIs to perform tasks such as image classification, object detection, text analysis, and more. Core ML handles the complexities of model execution, memory management, and hardware acceleration, allowing developers to focus on building innovative features for their users.

Machine Learning and Models

Machine learning is a branch of artificial intelligence that focuses on creating algorithms and models that can learn patterns and anticipate outcomes from data without being explicitly programmed.
In supervised learning, the model is trained using labelled data, with each input assigned a matching output. This lets the model to understand the relationship between inputs and outputs and make predictions on previously unknown data.

An image classifier model is a form of supervised learning model that assigns pictures to predetermined classes or categories. It learns to recognise patterns and characteristics in photos and classifies them based on their similarity to training samples.

Image Classifier Model

It works by analyzing the features and patterns present in images and using them to make predictions about the class of new, unseen images.

Training Phase

Data Collection: The first step in training an image classifier model is to gather a dataset of labeled images. Each image in the dataset is associated with a class label, indicating the category it belongs to. For example, if building an image classifier to distinguish between cats and dogs, the dataset would consist of images of cats labeled as "cat" and images of dogs labeled as "dog."

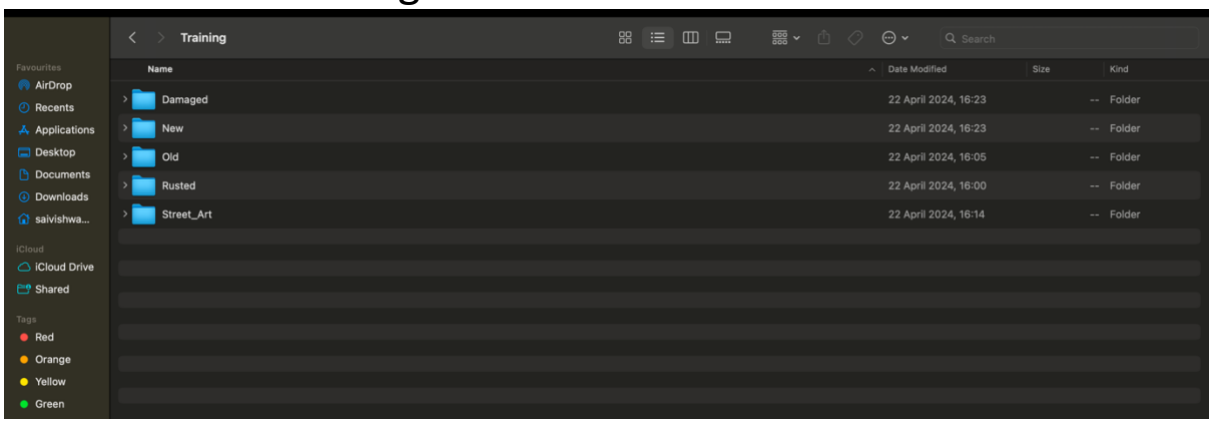
Data Preprocessing: Once the dataset is collected, preprocessing steps are applied to prepare the images for training. This may include resizing the images to a uniform size, normalizing pixel values, and augmenting the dataset with techniques like rotation, flipping, or cropping to increase variability and improve generalization.

Model Training: The dataset is split into training and validation sets. The image classifier model is then trained on the training set using algorithms such as convolutional neural networks (CNNs).

Loss Calculation and Optimization: During training, the model's performance is measured using a loss function, which quantifies the difference between the predicted labels and the true labels in the training data. The model's parameters are adjusted iteratively through optimization algorithms such as gradient descent to minimize the loss and improve accuracy.

Validation and Tuning: After each training iteration (epoch), the model's performance is evaluated on the validation set to monitor for overfitting or underfitting. Hyperparameters such as learning rate, batch size, and network architecture may be adjusted based on validation performance to improve the model's generalization ability.

Model Evaluation: Once training is complete, the final trained model is evaluated on a separate test set to assess its performance on unseen data. This ensures that the model can generalize well to new images and accurately classify them into the correct categories.

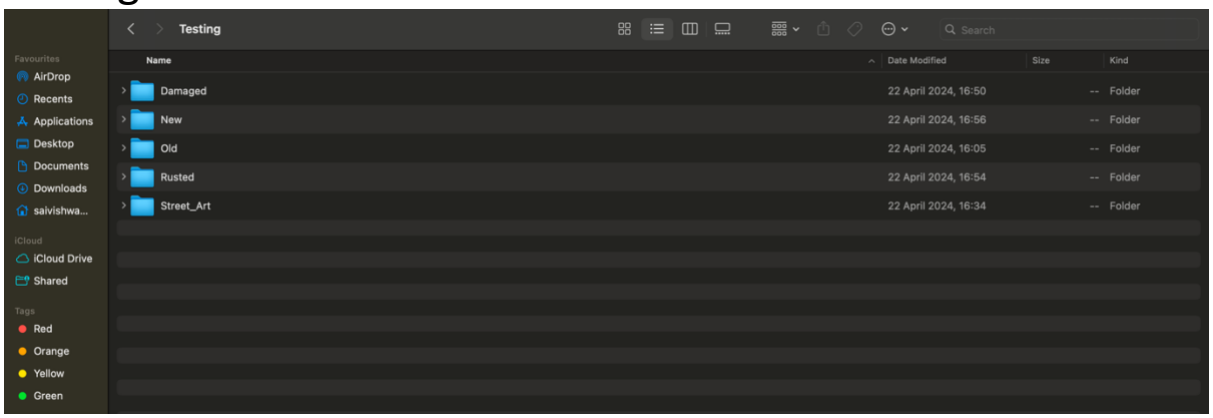


Testing Phase

Data Preparation: Similar to the training phase, the test dataset is preprocessed to ensure it's in the correct format and contains images representative of real-world scenarios.
Model Inference: The preprocessed test images are fed into the trained image classifier model for inference. The model predicts the class labels for each image based on the learned features and parameters from the training phase.
Performance Evaluation: The predicted labels are compared to the true labels in the test dataset to measure the model's performance metrics such as accuracy, precision, recall, and F1-score. These metrics provide insights into how well the model generalizes to new, unseen data and can help identify areas for improvement.
Iterative Improvement: If the model's performance is not satisfactory, further iterations of training, validation, and

tuning may be conducted to refine the model and enhance its accuracy and robustness.

Testing Data Set



App Functionality

The software utilizes Core ML to differentiate between different train boxcar conditions based on the degree of rust, the general structure, and appearance. The app classifies the railroad boxcars into five different conditions:

New: Railroad boxcars that have no rust and are brand new.
Rusty: Railroad boxcars which contain a lot of rust.
Old: Railroad boxcars that are too old.
Damaged/worn-out: Railroad boxcars which are completely damaged.
Street Art: Railroad boxcars that contain street art.
How the App is Created

The app is created using a main storyboard containing a navigation controller and a view controller with an image view. It also includes a navigation bar button item labeled "camera."

The app utilizes UIImagePickerControllerDelegate and UINavigationControllerDelegate to capture images from the user's device. It leverages the Core ML framework to interact with a custom image classifier model.

The following Core ML components are utilized:

VNCoreMLModel
VNCoreMLRequest
VNImageRequestHandler
Once an image is fed into the Core ML engine with the trained .mlmodel, it produces VNClassificationObservation as a result. This result contains confidence values for the input image, and the image with the highest confidence value is selected.

Core ML Limitations

Despite being advantageous for the iOS app developer community, Core ML has its limitations. The pre-trained model remains static, meaning it cannot be further trained using user-generated data. The model only learns what it was trained on initially and cannot be improved using additional data collected from users.