Divya Sravani Bandi
Mobile Application Development
Grad Project

Report on CoreML

Introduction:

Core ML provides the integration of machine learning models into iOS and macOS applications, enabling developers to create intelligent apps capable of solving complex problems. Core ML improves app intelligence by integrating trained machine learning models, enabling apps to perform sophisticated tasks such as image recognition, natural language processing, and predictive analysis. This integration allows developers to create intelligent apps capable of understanding and interpreting complex data.

Optimized Performance:

Core ML is made to work fast and smooth on Apple devices. It uses special hardware like the Neural Engine in Apple's A-series chips to make machine learning models run quickly and efficiently. This means that apps can quickly analyze data and respond in real-time, making the user experience better and more responsive.

Core ML Overview:

Core ML acts as the backbone for integrating trained machine learning models smoothly into applications. It supports different types of models, such as neural networks, tree ensembles, and generalized linear models. Core ML models are optimized for mobile devices, guaranteeing effective performance on Apple platforms.

Simplified Development Process:

Integrating machine learning models into iOS and macOS applications with Core ML is straightforward and uncomplicated. Developers can use pre-trained models from popular frameworks like TensorFlow or PyTorch, or they can train their own models using tools like Create ML. Core ML handles the complexities of model deployment and execution, providing a user-friendly interface for adding machine learning capabilities to apps.

On-Device Intelligence:

One of Core ML's significant advantages is its support for on-device machine learning inference. By conducting inference directly on the user's device, Core ML ensures data privacy and security

while reducing latency associated with cloud-based solutions. This allows apps to offer intelligent features even without an internet connection, enhancing user privacy and data efficiency.

Extensive Model Compatibility:

Core ML supports a wide range of machine learning model types and formats, including neural networks, tree ensembles, and linear models. This flexibility enables developers to select the most suitable model architecture for their specific use case and integrate it into their app. Additionally, Core ML provides tools for converting models between different formats, ensuring compatibility with the Core ML runtime.

Continuous Learning and Improvement:

With Core ML, developers can easily refine and enhance machine learning models to boost their performance over time. By gathering user feedback and training new models based on real-world data, apps powered by Core ML can continually learn and adapt to evolving user requirements and preferences. This iterative approach to model development enables apps to remain relevant and competitive in a rapidly changing landscape.

Training the Model:

To train a machine learning model, in this scenario, for predicting rust and corrosion levels on boxcars, several essential steps are involved:

Data Collection: Gathering a varied dataset of labeled boxcar images, representing different rust and corrosion levels.

Data Preprocessing: Preprocessing the dataset to standardize image sizes, formats, and quality, ensuring consistent training data and improved model performance.

Model Selection: Choosing an appropriate model architecture suited for image classification tasks, typically Convolutional Neural Networks (CNNs) due to their effectiveness in extracting features from images.

Training Process: Training the selected model on the labeled dataset using techniques like gradient descent and backpropagation. The model learns to identify patterns and correlations between image features and rust/corrosion levels.

Core ML Integration:

Integrating the trained model into the app using Core ML involves the following steps:

Model Conversion: Converting the trained model into the Core ML format (.mlmodel) compatible with iOS and macOS platforms, often done using tools like Core ML Tools or third-party converters.

Model Deployment: Embedding the converted Core ML model file into the Xcode project of the app, integrating it into the app's workflow.

Model Inference: Utilizing the Core ML framework to perform inference, i.e., making predictions, using the trained model on new input data. Core ML handles the complexities of model execution, ensuring efficient and optimized performance on Apple devices.

Prediction Process:

When a user uploads an image of a boxcar in the app, Core ML executes the following steps to predict rust and corrosion levels:

Image Preprocessing: Preprocessing the uploaded image, including resizing, normalization, and other transformations, to prepare it for input to the model.

Model Inference: Passing the preprocessed image through the trained Core ML model to obtain predictions. The model analyzes the image features and generates predictions based on its learned parameters.

Result Interpretation: Interpreting the model's predictions to determine the level of rust and corrosion on the boxcar. Predictions may be categorized into different levels or classes representing varying degrees of rust and corrosion severity, in the current scenario, best, bad, good, average and worst.

Displaying Results: Presenting the prediction result on the app's interface for the user to visualize and interpret. This could include displaying the predicted rust and corrosion level alongside the uploaded image for easy comprehension.

Conclusion:

Core ML empowers developers to create intelligent apps capable of solving real-world problems, such as predicting rust and corrosion levels on boxcars. By leveraging Core ML's capabilities in

model integration and execution, developers can deliver sophisticated machine learning experiences to users, enhancing app functionality and user engagement.