Bit-by-Bit : The Side Channel Hackathon

Hackathon Challenge 2

ModelSpy: Identify the CNN model from Side-Channel CPU Traces

Objective

- Develop an algorithm to identify which CNN model is running on a shared cloud CPU using only CPU side-channel telemetry data.
- Use labeled performance counter traces from known CNN models to train a classifier.
- Extract meaningful statistical features from raw CPU telemetry traces for robust classification.
- Ensure the classifier is accurate, robust to noise and timing shifts, efficient for quick inference, and explainable.
- Target CNN models include ResNet, AlexNet, VGG, DenseNet, Inception V3, MobileNet V2, and ShuffleNet V2.

Overview

In multi-tenant cloud environments, CPUs are shared among multiple users, allowing subtle side-channel information from one user's workload to leak into others' performance monitoring data. This project leverages this phenomenon by analyzing CPU hardware telemetry—specifically counts of cache misses, branch events, and instruction cycles captured using Linux perf—to identify which CNN model is running on the shared CPU. This is a challenging classification problem due to noisy traces and timing shifts.

- Raw telemetry traces are preprocessed by cleaning, normalization, and converting to numeric formats.
- Features such as mean, standard deviation, min, max, and median are extracted from fixed-size chunks of traces.
- A Random Forest classifier is trained on these features with cross-validation to differentiate among known CNN models.
- New traces are predicted by extracting features and aggregating multiple chunk predictions for higher confidence.
- The project demonstrates leveraging hardware side-channel data for workload fingerprinting and CNN model identification under noisy cloud conditions.

Methodology

Step 1: Data Collection

- Downloaded a referenced dataset of labeled CPU performance counter traces for various CNN models from the server.
- Collected new side-channel telemetry traces using the Linux perf tool by monitoring CPU events like cache misses, CPU cycles, branches, and branch misses.
- Traces were recorded under realistic cloud-like noisy conditions to simulate multi-tenant CPU usage.

Step 2: Training the Model

- Loaded the labeled reference dataset and pre-processed it by splitting the traces into fixed-size chunks (100 samples).
- Extracted statistical features from each chunk: mean, standard deviation, minimum, maximum, and median of the counts.
- Divided the labeled data into training (80%) and verification/test (20%) subsets.
- Trained a Random Forest classifier on the extracted features from the training data.
- Verified model accuracy on the 20% held-out data, achieving approximately 97% accuracy.
- Saved the trained model to a file for later use in predictions.

Step 3: Pre-Processing of New Traces

- Downloaded new unlabeled traces from the server for prediction.
- Cleaned the new traces by removing comments and bad lines, handling missing data, and normalizing count values between 0 and 1.
- Saved cleaned traces into an output folder to standardize inputs for feature extraction and classification.

Step 4: Prediction and Model Comparison

- Extracted features from the pre-processed new traces using the same chunking and statistical extraction method as used in training.
- Used the saved Random Forest model to predict the CNN model corresponding to each chunk in the new traces.

Analysis Technique

- Utilized statistical feature extraction from CPU side-channel telemetry traces to convert raw time-series data into meaningful numeric features.
- Extracted features included mean, standard deviation, minimum, maximum, and median values from fixed-size chunks of the recorded performance counters.

- Selected these statistical measures as they capture the distribution and variability patterns inherent to each CNN model's CPU execution profile.
- Employed a Random Forest classifier, an ensemble learning method effective at handling structured tabular data and discerning complex feature interactions.
- Used stratified cross-validation during training to ensure balanced representation of all CNN classes and robust performance evaluation.
- Applied per-chunk predictions and then aggregated predictions over entire runs to improve classification confidence and reduce the impact of noise and timing shifts.
- Visualized performance through confusion matrices and classification reports to analyze model accuracy and misclassifications.
- Normalization and cleaning of raw traces ensured consistency and comparability in feature space, thereby improving classifier reliability.

Results

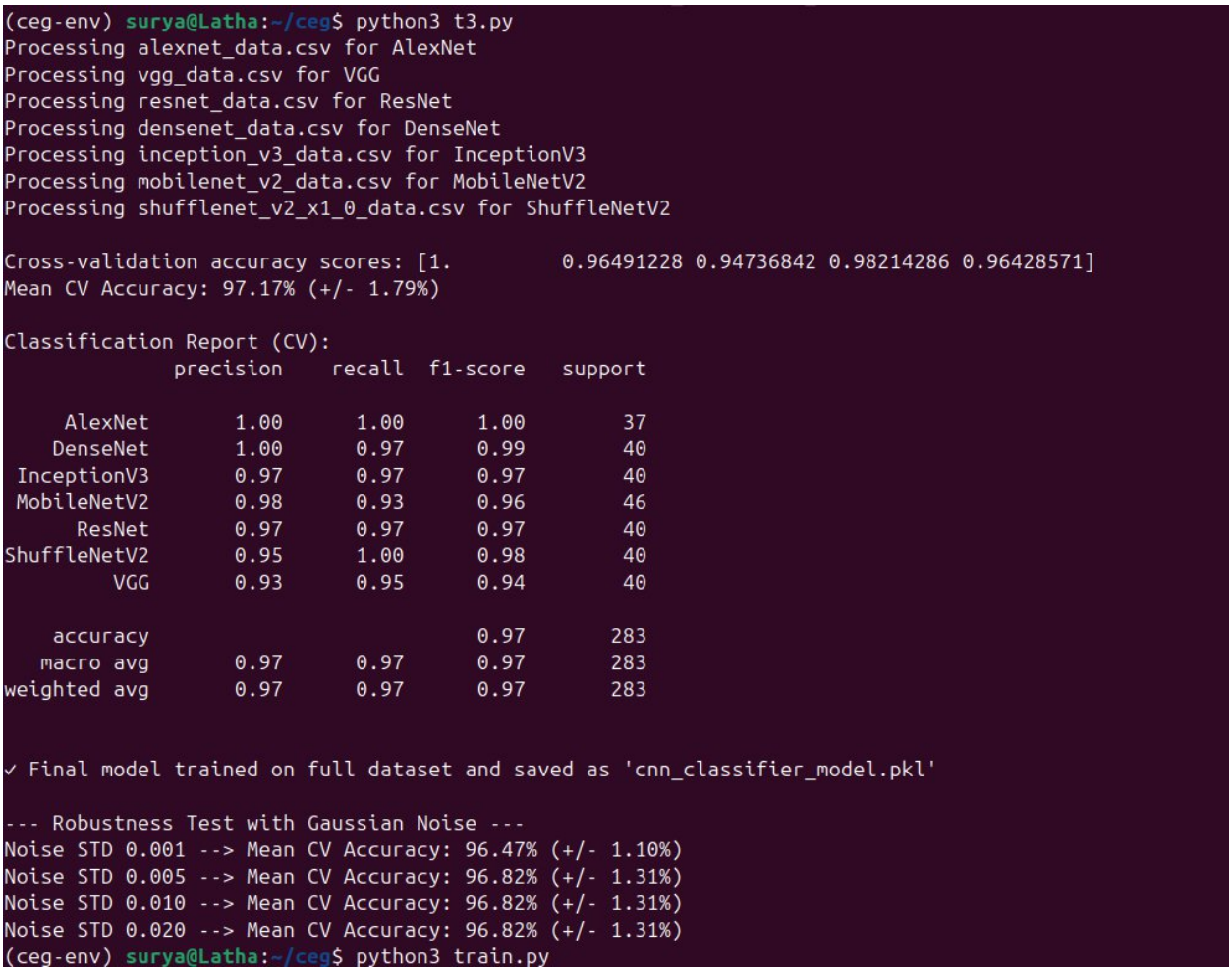

Figure 1. Training Accuracy

- We achieved an accuracy of 97% using the Random Forest method, both with and without the addition of Gaussian noise. Figure 1 illustrates the accuracy results obtained from our experiments.
- Using this accuracy, we predicted the CNN model running in the hackathon user's environment, as illustrated in Figure 2.

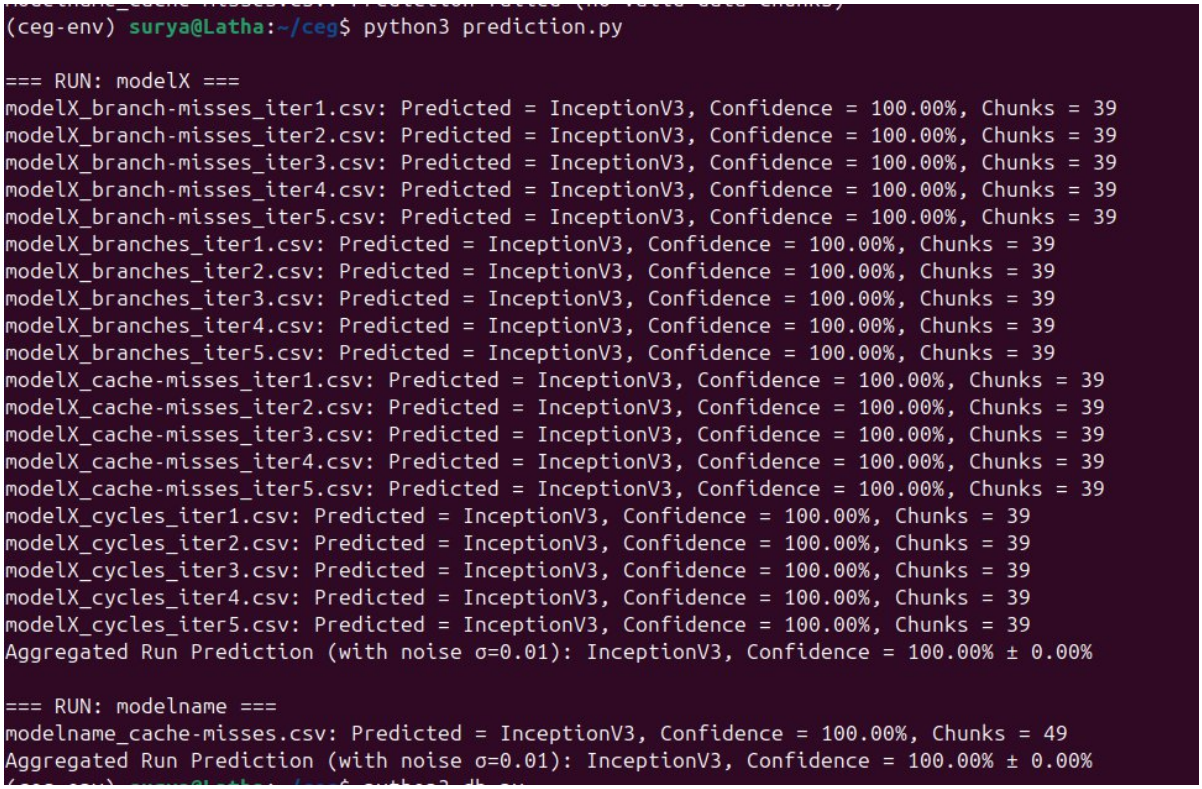

Figure 2. Predicted Model

Challenges Faced

- Limited labeled data made it difficult to train a generalizable CNN model classifier.
- Designing effective features and using robust cross-validation helped overcome limited training examples.
- Side-channel CPU data was noisy due to background processes and cloud system jitter.
- Variability in timing and trace length complicated feature extraction and required normalization techniques.

Conclusion

- This project validated that side-channel CPU telemetry traces can be effectively used to identify CNN models running in a multi-tenant cloud CPU environment.

- By preprocessing raw performance counter data and extracting statistical features, a discriminative representation of different CNN workloads was obtained.
- Training a Random Forest classifier on these features achieved a high accuracy of 97%, demonstrating strong model differentiation.
- Prediction results showed consistent identification of the InceptionV3 model with 100% confidence across multiple noisy traces and event types, proving the robustness of the approach.
- The approach highlights the potential for hardware side-channel analysis to serve as a non-intrusive fingerprinting method for workload monitoring and security in shared computing resources.
- Challenges such as noise, timing variability, and limited labeled data were effectively mitigated with normalization, chunk aggregation, and cross-validation.
- Future work could explore deep learning techniques, real-time detection, and explain ability to enhance accuracy and applicability in dynamic cloud environments.

The scripts are available at: https://github.com/suryaarasu11/bitbybit/