COURSE PROJECT REPORT

Efficient Vehicle Plate Recognition

Github Repo Link:
https://github.ncsu.edu/ambhansa/LPRNetOptimization_RealTimeML-AI.git

Q1. What is the original DNN model you chose to use?

The original DNN model used for the project was LPRNet. It is a high performance and lightweight license plate recognition framework. This model is used mainly for license plate recognition tasks. The model performs detection and character recognition in a single forward pass. It aims to be lightweight hence usable in resource constrained and real time environments. The model takes images of license plates as input and outputs recognized license plates characters. This model can prove to be useful in scenarios like traffic management, parking lot management etc.

Pretrained LPRNet model from this repo was utilized, its accuracy is approximately 90%, test speed is 0.232s and the model size (in onnx) is 1.91MB. Considering only the non zero parameters, the (onnx) model size of this original model is 1.11MB.

Q2. What optimizations did you apply to the model to improve its accuracy, speed and space efficiency?

To improve the accuracy, speed and space efficiency of the model, two model optimizations and MLC optimizations were applied. Each of these contributed to improve either speed, accuracy, or space or a combination of these:

MODEL OPTIMIZATIONS:

1. **Pruning:** Unstructured Pruning was applied. Pruning ratios ranging from 0.1 to 0.9 were applied and tested to check for the most efficient pruning ratio. Pruning ratio 0.5 gave the optimal balance between accuracy and efficiency. While accuracy dropped to about 82.3%, the inference time was reduced to 0.143s. The model size was also reduced to 0.71MB (considering only the non zero parameters.)

2. **Quantizaion:** Static quantization was used as another model optimization. Using dynamic optimization would not have given much improvements as it is inapplicable in this case because of absence of linear layers. Therefore static post training quantization (PTQ) was applied. This method reduced the precision

of model's weight and activations. It converts these from 32 bit floating point to 8 bit integers.
A representative dataset for calibration before finalizing the conversion was made. The fbgemm backend that was utilized, which is optimized for CPUs.
This enabled efficient quantized operations. This approach also gave reduction in size and speed improvements. In addition to these advantages, the approach also does not need retraining as it is a "post training optimization technique". However accuracy was reduced when quantization was applied.

When this approach was directly implemented on the original pretrained model, the test accuracy dropped to 74.3%, while the inference speed got more efficient to 0.028 seconds with its model size reduced to 0.51MB.

Machine Learning Compilation (MLC) OPTIMIZATIONS:
Apache TVM's autoschedular was used to optimize on compiler level. The autoschedular generated 23 optimizations. A total of 150 trials were conducted to explore the optimization space and identify the most effective configurations for execution.

Each trial involved generating and testing a different set of configurations (also known as schedules).Increasing the number of trials beyond 150 (e.g., to 200 or more) could uncover additional high-performing schedules, potentially leading to further improvements in performance.

The autoscheduler generated **23 unique optimizations** as a result of these trials. These optimizations were tailored to the specific characteristics of the LPRNet model and the hardware it was deployed on. The optimizations targeted critical aspects such as operator fusion, memory access patterns, and computational loops.

The key optimizations applied include:

1. **Operator Fusion**
    - Combines multiple operations (e.g., convolution, activation) into a single kernel, reducing intermediate data storage and memory access overhead.
    - Speeds up computation by minimizing the number of kernel launches and memory accesses.
2. **Data Layout Transformation**
    - Rearranges data in memory to match the hardware's preferred layout, enhancing memory access efficiency.
3. **Loop-Level Optimization**
    - Reorganizes loop execution to reduce computation overhead.

- Techniques include loop unrolling, tiling, and vectorization, which improve cache utilization and parallelism.

4. **Reduction Optimization**
   - Optimizes operations that involve reducing dimensions, such as summations or pooling.
   - Eliminates redundant calculations and improves computational efficiency.

While the MLC optimizations do not make a difference on the model size and introduce some compilation complexity, they significantly speed up with least possible reduction in accuracy as compared to other methods tried. The testing accuracy came out to be 89% and the inference speed was 0.047 seconds.

COMBINATION OF OPTIMIZATIONS:
The above optimizations were applied individually on the pretrained LPRNet model. In addition to this several combinations of these optimizations were also done and compared. In all, following optimizations and their combinations were carried out.

1. Unstructured Pruning on original model
2. Post Training Quantization on original model
3. MLC Optimization of original model
4. Post Training Quantization on the pruned model
5. MLC Optimization of pruned model

It was noted that Post Training Quantization on the pruned model gave least training accuracy of 66.8 percent, proving to contribute the largest amount of accuracy drop. However, it also gave the best reduction in size and inference time as 0.51 MB and 0.0024 seconds respectively.

Along with this it is important to note that the combination of MLC Optimization of the pruned model gave an optimal balance of all the metrics. The test accuracy obtained from this model is approximately 80 percent and the inference time came out to be 0.039 seconds. The model size after this combination of optimization was 0.71MB.

Q3. What is the performance (the three metrics) of the original model and optimized model?
Following table shows the comparison and contrast among the optimized models and their combinations:

Model	Accuracy (%)	Inference Time (seconds)	Size (MB)
Original Model **(Baseline)**	90.0	0.2500	1.11(onnx)
Pruned Model	82.3	0.1430	0.71 (onnx)
MLC	89.4	0.0470	1.11 (onnx)
Pruned + MLC Optimized	80.0	0.0390	0.71 (onnx)
Quantized Model	74.3	0.0280	0.51
Pruned+Quantized Model	66.8	0.0024	0.51

Original Model

The original model serves as the baseline, offering the highest accuracy among all versions:

- **Accuracy:** 90.0%
- **Inference Time:** 0.250 seconds
- **Model Size (onnx, non zero parameters):** 1.11 MB

The baseline has highest accuracy among all the optimised model. However, its relatively higher inference time (0.250 seconds) and unchanged size limit its suitability for real-time or resource-constrained environments. While the original model sets the standard for accuracy, it lacks the efficiency needed for applications where speed and memory are critical, and hence may not prove to be efficient for resource constraint environments all the time.

Most Efficient Model (Balanced):Pruned + MLC Optimized Model

The Pruned + MLC Optimized Model that combines pruning with MLC optimizations, reduces the size of the model, as well as improves computational and memory efficiency. It **can be considered the most efficient model:**

- **Accuracy:** 80.0%, showing a trade-off but still reasonably high for many practical applications.
- **Inference Time:** 0.039 seconds, one of the fastest among all models, making it highly suitable for real-time tasks.

- **Model Size:** 0.71 MB, significantly reduced compared to the original model (1.11 MB), which is ideal for deployment in resource-constrained environments like mobile devices or embedded systems.

About it's Efficiency:

- **Balance Across Metrics:**
    - The **accuracy** is lower than the original and MLC Optimized models, but still sufficient for use cases where some tolerance for error is acceptable.
    - The **inference time** is reduced to 0.039 seconds, making it faster than most other models, including the standalone MLC Optimized Model.
    - The **model size** is significantly smaller than the original and MLC-only models, enabling easier deployment in environments with memory or storage limitations.
- **Combination of Techniques:**
    - **Pruning** removes redundant connections in the neural network, reducing size while maintaining essential model performance.
    - **MLC Optimization** fine-tunes the model at the compiler level, enhancing computational efficiency and ensuring that the reduced size doesn't compromise speed.
- **Real-World Suitability:**
    - For applications where some accuracy trade-off is acceptable (e.g., high-speed systems or edge devices), this model delivers the best mix of compactness, speed, and reasonable accuracy.

Other Models May also be considered efficient based on resource availability and requirements:

1. **Original Model:**
   The original model is the most accurate (90%) but slower (0.250 seconds) and larger (1.11 MB). It's best suited for applications where precision is critical, and computational resources are abundant, such as backend systems or cloud-based processing.
2. **Pruned Model:**
   The pruned model is smaller (0.71 MB) and faster (0.143 seconds) but sacrifices accuracy (82.3%). This is a good choice for scenarios where memory is constrained, and a moderate drop in accuracy is acceptable.
3. **MLC Optimized Model:**
   The MLC optimized model retains high accuracy (89.4%) and significantly improves inference time (0.047 seconds). While it maintains the original size (1.11 MB), it's ideal for real-time systems that can tolerate the larger size.

4. **Quantized Model:**
   The quantized model achieves the smallest size (0.51 MB) and fastest inference time (0.028 seconds) but suffers from a steep accuracy drop (74.3%). It is efficient for ultra-lightweight applications where speed and size are more critical than precision.
5. **Pruned + Quantized Model:**
   This model is the smallest (0.51 MB) and fastest (0.0024 seconds) but has the lowest accuracy (66.8%), making it suitable for highly resource-constrained environments where even minimal accuracy suffices.

Why Pruned + MLC Optimized Strikes the Perfect Balance:
The model stands out because it effectively balances **accuracy, speed, and size**:

- It offers **reasonably high accuracy (80.0%)**, which is sufficient for many applications.
- It achieves **one of the fastest inference times (0.039 seconds)**, making it ideal for real-time use cases.
- Its **compact size (0.71 MB)** enables deployment in memory-constrained environments while maintaining performance.

This combination ensures that the model is versatile and practical across a wide range of scenarios, providing a middle ground between models optimized for accuracy, speed, or size alone.

Q4. What lessons have you learned through the project?
Following are the lessons I have learned through the project:

1. **Balancing the Metrics for Real-World Deployment**
   Through model optimization, I realized the critical importance of balancing key metrics like accuracy, inference time, and model size. These factors are essential when considering deployment in real-world applications, where each metric impacts the overall performance and feasibility of the model.
2. **Effectiveness of Optimization Techniques**
   The use of pruning proved to be a valuable technique in reducing the size of the neural network and improving efficiency by removing redundant parameters. Interestingly, this process didn't significantly harm the model's performance when the right pruning ratio in the right pruning technique is chosen, which showcased the effectiveness of pruning in optimizing deep learning models.
   MLC optimizations also highlighted the importance of fine-tuning at the compiler level. By applying these optimizations, I was able to significantly enhance computational efficiency without always sacrificing accuracy, which is important while deploying models in resource-constrained environments.

3. **The Power of Iterative Experimentation**
   One of the major takeaways from the project was the importance of iterative experimentation. I tested various optimization techniques such as pruning ratios and quantization levels. This iterative process allowed me to refine the model configuration and identify the most effective approach for different use cases. It reinforced the need to continuously experiment and evaluate different methods to achieve the best balance of performance.
4. **Understanding Trade-offs**
   The project reinforced the concept of trade-offs in machine learning. Improving one aspect of a model, whether it's size, speed, or accuracy, often comes at the expense of another. This project allowed me to better understand how to manage these trade-offs and make informed decisions based on the specific requirements of the task at hand.
5. **Adapting to Deployment Needs**
   A key lesson was the importance of tailoring model optimizations to fit the deployment scenario. Whether it's for edge devices with limited resources or cloud-based systems with more computational power, optimizing a model for its deployment environment is crucial. The project provided valuable insights into how deployment constraints influence the model selection and optimization process.
6. **Using Advanced Tools and Frameworks**
   Working with advanced frameworks like Apache TVM and leveraging techniques such as post-training quantization helped me understand the cutting-edge tools available to enhance model performance. These tools significantly improved the efficiency and computational speed of the models without compromising too much on accuracy.
7. **Critical Thinking for Model Deployment**
   This project helped me develop a deeper understanding and the ability to convert my theoretical knowledge to how it can be used practically for real applications. It encouraged me to think critically about the suitability and impacts on the different models of the optimization techniques. Understanding the real-world implications of different optimization strategies has now given me an experience and perspective to make informed decisions regarding model deployment.

Overall, this project has given me a fresh perspective, especially on the importance of model metrics such as accuracy, inference time, and model size while deploying models in real-time scenarios. The lessons I've learned about optimizing models for practical use and navigating the trade-offs involved in model performance will guide me in future projects, especially when working with models designed for resource-constrained environments or real-time applications.