CIFAR100 Convolutional Model Based Classification Benchmark

Adithya Balaji
Department of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
abalaji3@andrew.cmu.edu

Yuning Wu
Machine Learning Department
Carnegie Mellon University
Pittsburgh, PA 15213
yuningw@andrew.cmu.edu

Junwoong Yoon
Machine Learning Department
Carnegie Mellon University
Pittsburgh, PA 15213
junwoony@andrew.cmu.edu

1 Introduction

The problem we choose to solve is the classification task on CIFAR100 dataset (MP-B). Our motivation is to get a sense of how traditional machine learning (ML) methods, such as logistic regression (LR) and gradient boosted trees (GBT), compare to deep learning (DL) methods, such as convolutional neural networks (CNN), in complex computer vision (CV) problems. Besides the comparison, we'd also like to explore the possibility of combining the two methods (traditional ML and DL), and see whether that could incur a better performance. Despite the success or failure of such trial, we wish to gain a more detailed insight into how different models are performing with respect to different image classes, which classes have good performances & which have bad, and what might be the underlying reason of such performances. Through these analyses, it is our goal to obtain a better understanding of the selection, pipeline, cost and performance of developing practical ML / DL models on large datasets.

CIFAR100 dataset was developed to benchmark computer vision classification algorithms. [3] The dataset contains 60,000 32x32 images with 50,000 training images and 10,000 test images. As the dataset name suggests, the CIFAR100 contains 100 classes (as 'fine' labels) of objects group, evenly across 20 super-classes (as 'coarse' labels) that are distributed across the dataset resulting in 600 images (500 train + 100 test) per class. Additionally, it is important to note that this dataset does not contain any ambiguity in terms of multiple classes of objects in the same dataset. We used fine labels in our problem setting.

2 Methodology

Fig.1 depicts our pipeline. We used `torchvision.datasets` to obtain the CIFAR100 dataset. It was then preprocessed with steps such as visualization, preview and normalization. We first used it to train and tune the pure CNN model. Network architecture (see Fig.4 in Appendix) is fixed during this process, and we used Adam as the optimizer, cross-entropy loss as the loss function. [2] We evaluated performance of the model and tuned the hyperparameters with cross-validation. We saved the best model and its parameters. This CNN model is then used for feature extraction in training LR and GBT models. Similarly, we experimented with the hyperparameters, but mainly use stochastic gradient descent (SGD) and batch gradient descent as optimization methods. After training and evaluation, we post-process our models with visualization, class-wise analysis and model-wise comparison. Finally, we conclude our findings with numerical results, visual outputs, and heuristic analysis.

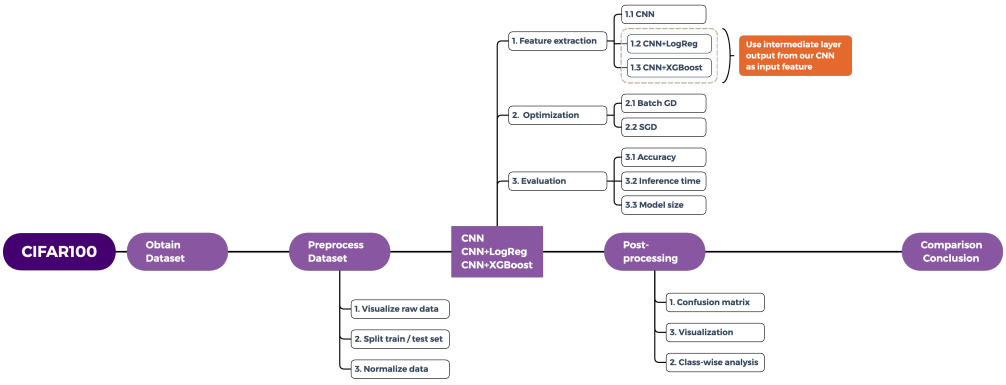

Figure 1: Our pipeline

We fully experimented on three models in total, one is a pure CNN architecture, the other two are models combining CNN and traditional ML, namely CNN+LR and CNN+GBT. We took this approach after an initial survey which found the top-1 accuracy of a pure LR or GBT model on flattened images to be <1%. In the combination, we use intermediate layer output of our CNN model as input features for logistic regression / GBT, which can essentially be seen as a feature extraction procedure. We deem these techniques suitable for the CIFAR100 classification task based on the following considerations.

- For deep learning model, we choose CNN over RNN, because CIFAR100 is a large image dataset. Heuristically CNN is more suitable for image classification tasks, and have known to performed well in other CV challenges such as ImageNet. Also, there isn't a strong implication for sequential setting, so we did not choose RNN.
- For traditional ML model, we choose GBT because it's an optimized distributed gradient boosting method that usually out-performs other ensemble CART methods such as random forest. We consider it as the strongest traditional ML candidate for CIFAR100.
- We choose logistic regression because of its inherent capability of handling multi-class classification. Other models such as SVM are natural to handle binary classification, but will inevitably run into OvR (One-versus-Rest) or OvO (One-versus-One) comparison when expanded to multi-class setting, which can be less efficient in CIFAR100's case.

For evaluation metrics, We use accuracy as the major performance score. We report both Top-1 and Top-5 accuracies to demonstrate the performance of our classification models. Our models are trained with cross-entropy loss and the loss is also reported as one of the evaluation metrics. We additionally choose inference time per image sample, model size, total number of parameters as reflections of the models' time and storage cost. In later analysis, we also evaluate the difficulty of using different models.

To answer the questions we raised in part 1, we have put thorough consideration into whether or not we should be using pure LR / GBT in comparison with pure CNN. Our final decision was to not use them because of the following reasons.

- From our previous knowledge of logistic regression and GBT, their direct application on classification tasks as challenging as CIFAR100 may only yield terrible results (<10% accuracy) that are not even comparable to CNN's result.
- **Feature extraction is an important link in the machine learning pipeline.** Using pre-trained CNN not only helps with this link, but may also help with answering our question of "whether combining traditional ML and DL can incur better performances".

The reason we did not use other well pre-trained CNN models such as VGG19 for feature extraction, is to establish a unified scenario for model evaluation and comparison. So that we know any performance difference comes from selection of model, not from other factors or boosts.

3 Computation

The initial plan was to use AWS as the cloud provider and SageMaker as the tool of choice to put each of these models to the test. Specifically, we were looking to use EC2 P3 instances which are equipped with Tesla V100 GPUs. After some initial experimentation, we found that this setup was actually quite unnecessary for our given dataset as we had overestimated the amount of time that the model would take to train with the CIFAR-100 dataset. We ended up using a combinations of our local compute resources (i.e. desktop machines) and Google Colab to train and evaluate our models.

Each of us used our local machine to train the models before post-processing. As described in the pipeline, CNN model is trained first with a computer equipped with 11 GB Geforce RTX Ti 2080 GPU and Intel Xeon CPU with 2.40 GHz clock speed. Using this saved, pre-trained model for feature extraction (pure inference), LR and GBT are then trained with a computer equipped with a Titan Xp GPU with 3840 CUDA cores and 12GB of GPU memory space. The CPU for the latter models is Intel Core-i7 CPU with 3.7GHz clock speed and 12 CPUs.

For the final comparison, we used Google Colab as a shared environment to compare common metrics like inference time. The CPU is an Intel Xeon with a 2.30 GHz clock speed and has a single core with 2 threads. The GPU is a Tesla T4 with 320 tensor cores, 2560 CUDA cores, and 16 GB of GPU memory space.

All of the code was written in Python. The core CNN model was constructed using PyTorch. The Gradient Boosted Tree (GBT) model was constructed using `xgboost`[1]. The Logistic Regression (LR) model was constructed using `scikit-learn` [4]. In terms of infrastructure, as mentioned before, Google Colab was used as the computer service of choice. For plotting the charts, we used `tensorboard`, `matplotlib`, and `seaborn`. The number of dependencies for the project totaled 105. A preview of Python imports can be found in Appendix.

Since we chose to go with a free infrastructure solution, the training cost of the models was $0. In terms of training, all three of the models took roughly 1.5 hours since we were trying to control for computation time. Although, the Logistic Regression and Gradient Boosted Tree approaches had the added benefit of using the intermediate results of the CNN model. In a sense, you could add the CNN training time to the latter two models.

4 Results

In Table 1 we evaluate each of the proposed classification models on test set of CIFAR100 using five different metrics. We see that CNN performs the best in terms of the loss and accuracy. CNN reports the lowest cross-entropy loss (1.813) and the highest Top-1 (59.51%) and Top-5 (83.53%) accuracies. CNN+GBT performs slightly worse than the CNN, but it still outperforms CNN+LR. One possible explanation for this result is that the extracted features from the intermediate layer of CNN (Figure 4 in the Appendix) were updated in order to optimize the pure CNN model. According to Figure 2, the extracted features from CNN can pick up some patterns for each of the images but these features might not be optimal representations of the input images for the derived models.

Table 1: Performance of each model on different evaluation metrics.

Model	Loss	Accuracy (Top-1)	Accuracy (Top-5)	Inference Time (μs)	Size (MB)
CNN	1.813	59.51 %	83.53 %	283.53	63.67
CNN+GBT	1.947	51.98 %	77.68 %	608.29	69.51
CNN+LR	6.812	40.19 %	46.83 %	294.19	64.50

Table 2 shows the three best and worst performing classes for each model. Interestingly, we can see some of the classes are commonly found across different models. For example, orange is the first best performing class for CNN, but it is also one of the best performing classes for CNN+GBT. On the other hand, otter and seal are commonly appeared as the worst performing classes for all models. We believe this highly correlated results between CNN and other derived models are due to the use of the shared intermediate features extracted from CNN.

Fig.3 shows some of the wrong labels that our models misclassify as either otter or seal. Our models are often confused by images that contain water background and/or animals with tails, and find it

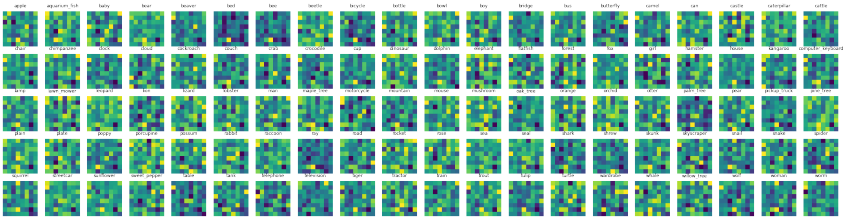

Figure 2: Visualization of CNN extracted features

Table 2: Three best and worst performing classes for each classification model. Top-1 accuracy is enclosed in parentheses. Commonly found classes are in bold.

Model	Best 1	Best 2	Best 3	Worst 1	Worst 2	Worst 3
CNN	**orange** (91%)	**road** (88%)	**wardrobe** (88%)	**otter** (24%)	**seal** (27%)	lizard (30%)
CNN+GBT	plain (84%)	**orange** (80%)	**wardrobe** (80%)	**otter** (14%)	**seal** (21%)	girl (22%)
CNN+LR	**road** (83%)	apple (83%)	cloud (83%)	**otter** (13%)	bear (21%)	**seal** (26%)

difficult to distinguish them from otter or seal. We also observe that the classification criterion of the three models rely heavily on contours and theme colors. As such, classes with simpler contours and more unified colors (such as orange) may yield better performances. More details about class-wise accuracy for each model can be found in Fig.5 in the Appendix. Despite differences in values, it is easy to detect a shared pattern / correlation among the models through this bar chart visualization.

Figure 3: Classes with worst accuracy, and the wrong labels they are assigned

We also compare the cost of the models based on the inference time and the model size in Table 1. Inference times of CNN and CNN+LR are very similar and they are about twice as fast as CNN+GBT. Actual sizes of the three models are not so different because the traditional ML methods (LR and GBT) are light compared to CNN. For ease-of-use consideration, CNN is the most challenging one. Due to its sophisticated architecture, it is more complicated to implement or use. LR and GBT, on the other hand, are easier to implement and use.

In summary, we reach the following conclusions corresponding to questions raised in part 1,

- Traditional ML methods perform worse than DL methods on CIFAR100 classification task, despite the feature extraction techniques employed.
- A combination of ML and DL methods will not essentially incur an improvement in performance. However, such combination may have the merit of a simpler usage as oppose to pure DL methods.
- Some class-wise similarities can be observed among the three models. Such correlation could be related to the dependency on CNN extracted features.
- Some visual patterns could be observed to partially explain the success and failure of the models.

References

[1] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: ACM, 2016, pp. 785–794. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939785. URL: http://doi.acm.org/10.1145/2939672.2939785.

[2] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[3] Alex Krizhevsky. "Learning Multiple Layers of Features from Tiny Images". In: *University of Toronto* (May 2012).

[4] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

Appendix

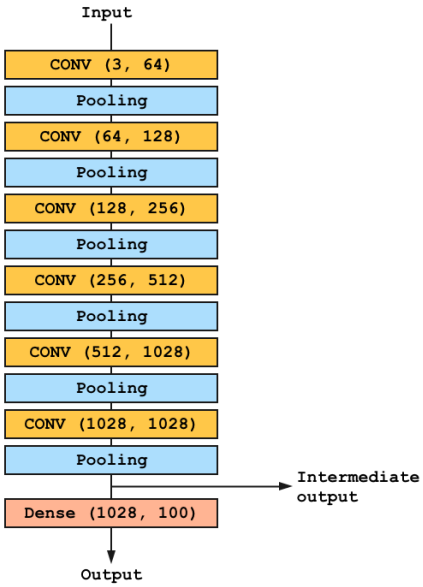

Figure 4: Our CNN Architecture

Preview of major Python libraries in use.

```
import torch
import torch.nn as nn
import torch.nn.functional as F

import torch.optim as optim
import torchvision
import torchvision.transforms as transforms

from sklearn.metrics import accuracy_score, log_loss, confusion_matrix
from sklearn.model_selection import learning_curve, ShuffleSplit

import matplotlib.pyplot as plt
import seaborn as sb
import numpy as np
import time
import pickle

# Specific to logistic regression
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import SGDClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline

# Specific to GBT
import xgboost as xgb
```

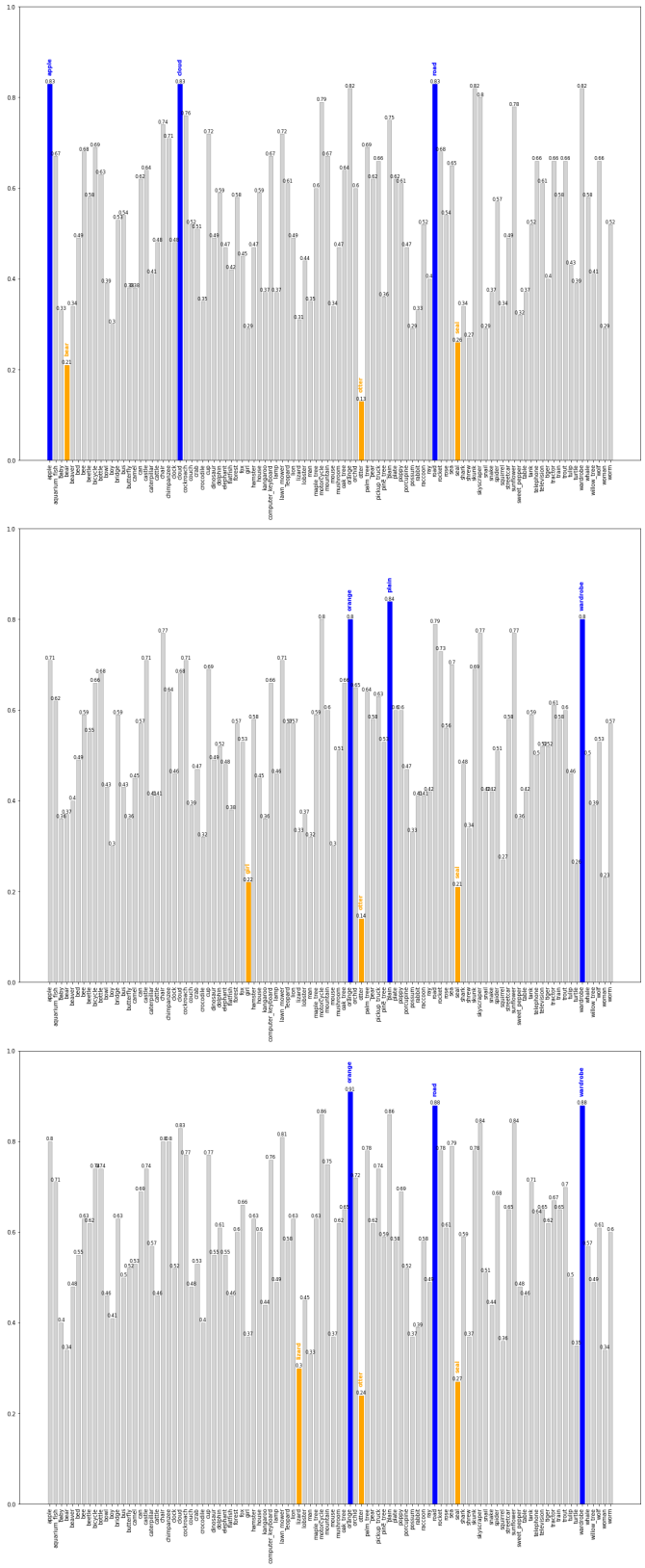

Figure 5: From top to bottom: class-wise accuracy for CNN, CNN+LR, CNN+GBT. Top 3 classes (in terms of accuracy) are marked in blue, and last 3 classes are marked in yellow.