Final Report for 10-805 Mini Projects

Weiran Yao
Carnegie Mellon University
Pittsburgh, PA 15213
wyao1@andrew.cmu.edu

Bingqing Chen
Carnegie Mellon University
Pittsburgh, PA 15213
binqinc@andrew.cmu.edu

Arnav Choudhry
Carnegie Mellon University
Pittsburgh, PA 15213
achoudhr@andrew.cmu.edu

1 Project A: Song Hotness Prediction on the Million Song Dataset

1.1 Introduction

Music has been a mainstay of human society since prehistoric times. It is found across cultures, and is used as a mode of expression, storytelling, and even recording and passing on knowledge. Commercially, the global music industry is valued at about $20.2 billion in 2019 [1]. A big part of this valuation is derived from sales and streaming of *popular* music. For a streaming platforms such as Spotify or Apple Music, or a recording labels such as Sony BMG or Warner Brothers, it would be very useful to know whether a song is going to be popular or not before publishing it. With this information, studios would be able to allocate resources such as song contracts more effectively, while also providing the users or listeners more music that they would like.

In this project, we use song hotness as a proxy of whether a song became popular or not. We use song metadata and acoustic features from a song to predict how popular (or hot) a song is going to be, by using linear regression. By solving this regression problem, we can find features which contain the most predictive power for predicting the popularity of a song. As a studio, we can then focus on these features while trying to figure out whether to sign on a particular artist or not. We will also study this problem in the context of large datasets by looking at space and time performance of implementing regression in a distributed setting. Our code for all models and experiments is available on GitHub[1].

1.1.1 Dataset

The Million Song Dataset (MSD) is about 280 GB in size, and was created for the purposes of research on algorithms at-scale and provide a standard dataset for various Music Information Retrieval (MIR) tasks [2]. It contains 1,000,000 songs from 44,745 unique artists. The data is stored as a million HDF5 files. Each song has one HDF5 file. For each song, MSD contains acoustic features such as duration, energy, loudness and metadata such as artist name, title, year of release. We use the variable "hotttnesss" as the target variable for the regression problem.

1.2 Methodology

Figure 1 illustrates the ML pipeline for this task, which consists of data loading and cleaning, data exploration, feature engineering, evaluation and model selection.

1.2.1 Get data

Data loading We convert MSD to standard CSV format and upload it to S3 in this step. The full MSD is available as Amazon EBS snapshot (snap-5178cf30) and can be attached to EC2 instance launched in us-east-1. We use **song-hotttnesss** field as the ground-truth label. The following fields are extracted as candidate features for further analysis:

[1] https://github.com/weirayao/10805-mp

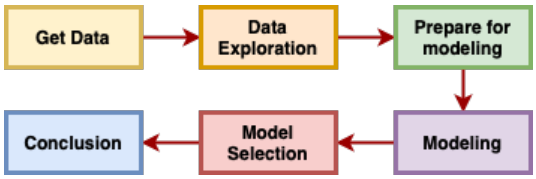


Figure 1: ML pipeline for mini project A.

- **Artist:** artist-id, artist-familiarity, artist-hotttnesss, artist-latitude, artist-longitude
- **Music:** danceability, energy, duration, start-of-fade-out, end-of-fade-in, key, key-confidence, loudness, mode, mode-confidence, tempo, time-signature, time-signature-confidence
- **Metadata:** title, genre, artist-terms, artist-terms-freq, artist-terms-weight, year

Data cleaning Song records whose **song-hotttnesss** is missing or has zero value are directly removed. We fill the missing values of text fields (e.g., title, artist-terms, etc.) with empty strings and fill the NAs of numeric fields (e.g., artist-familiarity) with 0. Two columns, which include danceability and energy, are removed because their values are zero for all songs in the dataset. The lat/long of artist location is spatially joined with continent shapefile and a new field "continent" is added. The lat/long fields are then removed. After data cleaning, 457,205 songs remain in the dataset.

1.2.2 Data exploration

We first conduct descriptive analysis on a subset dataset with 10,000 rows. The correlation of some numerical fields is visualized in Figure 2. We find that artist-hotttnesss and artist-familiarity are linearly correlated with song-hotttnesss. The mean value of song-hotttnesss is 0.49 and can be approximated as being normally distributed.

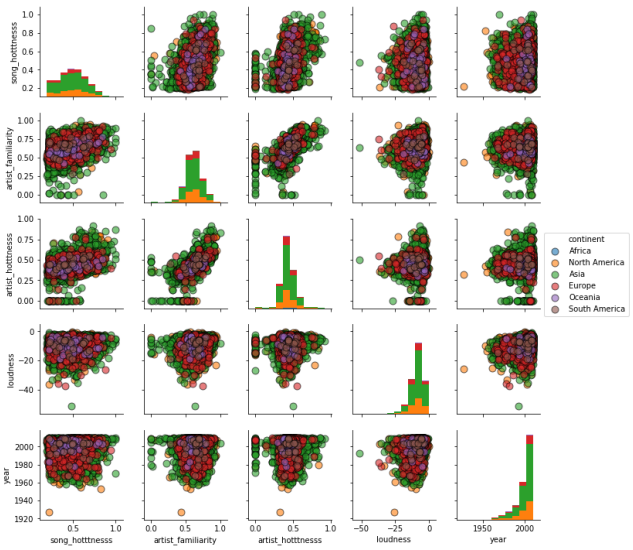


Figure 2: Pairplot of numerical columms of the subset dataset.

1.2.3 Feature engineering

For categorical variables including continent, key, mode, time-signature, we use one-hot encoding (OHE) to transform them into sparse vectors. For text variables including song title and artist terms, we use bag-of-words (BOW) encoding. Considering the large number of distinct categories by OHE and BOW, we also implement a variant which **hashes** the tuple (artist-id, title, artist-terms) with 100 buckets. For numerical variables, we subtract end-of-fade-in from duration to shift the feature. All numerical variables are transformed within 0-1 by Minmax normalization.

1.2.4 Modeling

We model the relationship between song hotness and artist and song information in linear regime. Considering the high-dimensionality and collinearity of feature set, two types of sparse linear models including Lasso and ElasticNet are implemented.

We do experiments with one baseline and four model variants.

Constant-baseline Baseline model which uses the average value of song-hotttnesss of training dataset for all predictions on testing set.

Lasso-ohe Lasso regression model which uses OHE/BOW to encode categorical and text variables;

Lasso-hash Lasso model which hashes the tuple (artist-id, title, artist-terms) into 100 buckets;

Elasticnet-ohe ElasticNet regression model which uses OHE and BOW to encode categorical and text variables. The ratio α between L_1 and L_2 is set as 0.5;

Elasticnet-hash ElasticNet model which uses feature hashing; The ratio is set as $\alpha = 0.5$.

1.2.5 Evaluation and model selection

Three evaluation metrics including **root-mean-squared-error (RMSE)**, **training time** and **memory usage** are considered. We run experiments with different regularization penalty λ and the best model in terms of RMSE on testing set is selected. The model weights are visualized and explained.

1.3 Computation

All data analysis and experiments were conducted on Amazon Web Services (AWS) platform. The raw data is saved as Amazon EBS snapshot.We used Amazon EC2 instances to extract, transform and load (ETL) the data into S3 storage device. The experiments and data analysis were run on Amazon EMR computing clusters.

1.3.1 Computing resources

We used one c5.4xlarge (16 cores, 32 GiB) attached with 500 GB volume to load the raw snapshot and transform it to CSV files on S3. The experiments were run on Amazon EMR computing clusters with 1 master node (m5.xlarge) and 2/5/20 core nodes (m5.xlarge) to evaluate the impacts of cluster configurations on the computational performances of ML pipeline. All instances are on-demand.

1.3.2 Environment setup

ETL pipeline The ETL pipeline used h5py to extract useful fields from HDF5 files and used pandas and boto3 to load the transformed data into S3 devices. joblib was used to parallel the ETL pipeline on 16 cores on the EC2 instance.

Experiments All models and experiments were implemented in Python and PySpark. The EMR cluster was created with emr-5.31.0 release, with Hadoop, JupyterHub, Spark installed for computation and with Ganglia and Zeppelin for resource monitoring. Pandas is also installed on the cluster through JupyterHub. Spark was configured with maximizeResourceAllocation option.

1.3.3 Challenges and cost

ETL pipeline We initially used the starter code on Piazza but the processing speed was too slow. Processing one chunk (1,000 HDF5 files) took 30 seconds on average and processing the entire dataset would take more than 8 hours. We thus chose to parallel the ETL pipeline with joblib and ran it on 16 cores. The total ETL process took 44 minutes.

ML experiments Because our feature set contains both dense and sparse vectors, when using vectorAssembler to convert it to LabelPoint object, the feature names were removed and cannot be recovered with MLlib. We thus chose to separate the ML pipeline into preprocessor and model. The preprocessor pipeline contains stringIndexers, oheEncoders + bowEncoders and hasher. The model pipeline contains vectorAssembler and the ML model. This also speeds up the training as the processed data can be reused for different models. In addition, we found that the results could not be saved locally on master node. We converted the Pandas dataframes back to spark dataframes and rdds, repartitioned them to 1 part and saved them directly to s3 through JupyterHub.

Total cost The S3 storage costs $0.85 per day. The ETL pipeline costs $1.61. Running all experiments on EMR cluster (3 cluster settings) costs $16.59. The total cost is $24.15.

Computation time ETL pipeline took 44 minutes. Running all experiments on EMR took 87 minutes. Note we ran experiments on three EMR cluster settings at the same time.

1.4 Results

From Figure 3 we can see that among the non-hashed features, the bag of words features created from the song title and artist terms are the most important. The familiarity of the artist and their hotness are also major deciders of whether a song is going to be hot or not. This is interesting since it seems that acoustic features don't seem to matter, as long as a famous artist releases a song with a good-sounding name, the song would probably be a hit. Of course, there are limitations on the data collection process. We can see that on hashing the features, they are not very interpretable anymore, and also give way to other features such as artist location as more important. The contribution of the individual hashed feature is also lower.

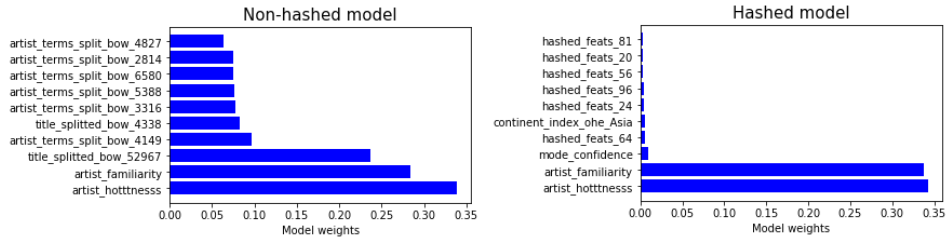


Figure 3: Top 10 features of the best hashed and non-hashed model.

We can see from Table 1 that hashed features do lead to some memory savings for the distributed memory usage of the spark dataframe. We utilize about 50% memory with hashed features than we do with hashed features. But the saving is not that much if we look at the scale of the numbers since both are using sparse vector representations.

Table 1: Distributed memory usage.

Model	**Memory usage**
Non-hased	167.66 MiB
Hashed	85.59 MiB

Figure 4 presents us with several expected and new results. First, we note that training time with hashed features is about constant irrespective of hardware or software configuration. This is as expected because there are only about 100 features after hashing. We also note that non-hashed feature sets perform better than hashed ones, at the expense of training time. This is also expected as the non-hashed features do not loose some information while hashing. Note that lower regularization leads to better models, which suggests that there is more training to do. However, hashed features seem to hit some sort of ceiling for improvement in performance. For the non-hashed features, we can see that lower regularization leads to more features, which leads to more training time. Going for a cluster size of 2 to 5 brings gains in training time, but going from 5 to 20 does not reduce training time by much, probably due to increase in communication time. LASSO seems to be taking a little

longer for training compared to ElasticNet, which might be due to the higher sparsity that LASSO searches for in the solution set.

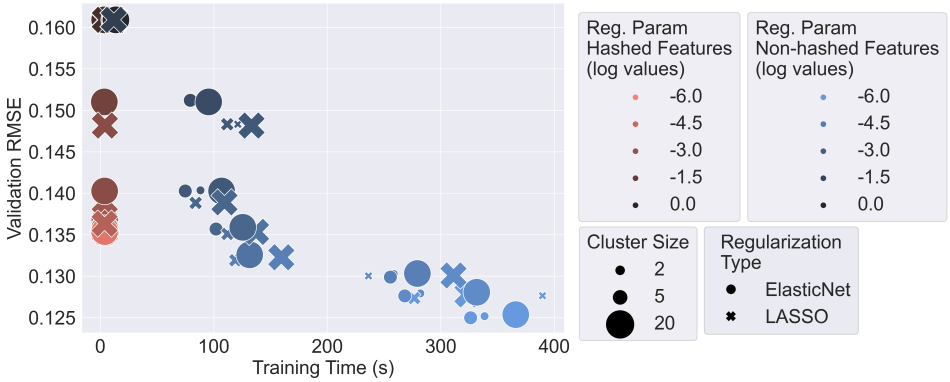


Figure 4: Performance versus training time of linear regression with varying degrees of regularization with ElasticNet and LASSO on different distributed computing setups.

2 Project B: Image Classification on CIFAR-100 Dataset

2.1 Introduction

Convolutional Neural Networks (CNNs) have achieved remarkable success in various computer vision tasks, e.g. image classification, semantic segmentation. While there is significant research on designing more accurate or more efficient models, there is limited research on analyzing and benchmarking their computation cost.

In this project, we use the CIFAR-100 dataset to understand the multi-way relationship between classification performance, model complexity and computational costs. We implemented three CNNs, plus one variant. While we initially considered simpler models, e.g. Logistic Regression and Random Forests, our analysis showed that they require a lot of feature engineering and do not do well in comparison to CNNs. Thus, we limit our experiments to CNNs.

We refer to the experimental design in [3] to study the relationship between classification accuracy, model complexity, and computational cost for different CNN models, as summarized in Figure 5. Our code for all models and experiments is available on GitHub[2].

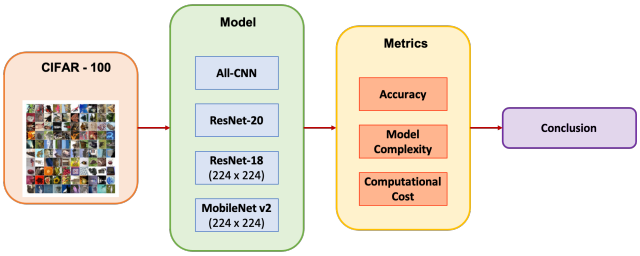


Figure 5: ML pipeline for mini-project B.

2.1.1 Dataset

The CIFAR-100 dataset [4] consists of 60,000 32×32 color images in 100 classes. The dataset occupies about 200MB of disk storage. There are 600 images for each class, which are split into 500 training images and 100 test images. Each image has the `coarse` (superclass) label and `fine` (class) label. We use the 100-class `fine` labels for this project.

[2] `https://github.com/weirayao/10805-mp`

2.2 Methodology

Our experimental design is largely inspired by [3], which benchmarked the computation cost of 40+ CNNs on two computing infrastructure using ImageNet [5]. Our mini-project can be seen as a reduced version of [3].

2.2.1 Models

We selected three CNN models that are all relatively light-weight.

All-CNN All-CNN [6], as the names suggests, replaces max-pooling layers with convolutional layers, resulting in an architecture consisting of only convolutional layers and ReLU activation. Specifically, we used the All-CNN-C variant in the paper.

Resnet-18 By introducing residual learning, the family of ResNets [7] address the challenge of diminishing gradients and unleash the potential of much deeper networks. ResNet-18 is the most light weight in the family.

MobileNet v2 MobileNet v2 [8] is an architecture tailored for mobile and resource constrained environments. It uses techniques such as depthwise separable convolution and inverted residual blocks to reduce model complexity and computation cost.

2.2.2 Metrics

Accuracy: We use top-1 accuracy to evaluate the classification performance on CIFAR-100. Accuracy is defined as the percentage of samples for which the classifier makes the correct predictions.

Model Complexity: We evaluate the model complexity with the number of learnable parameters. We implemented that with a simple function that adds up all the learnable parameters.

Computational Cost: We consider the GPU memory utilization and time cost during both training and inference. The GPU utilization is monitored via the `nvidia-smi` command, and computation time is evaluated as wall-clock time.

2.2.3 Implementation Details

Preprocessing For All-CNN, we normalize the dataset based both by sample and by feature. In order to use pretrained weights for Resnet-18 and MobileNet v2, we upsampled the images to 224×224 and normalized each channel with $\mu = [0.485, 0.456, 0.406]$, $\sigma = [0.229, 0.224, 0.225]$.

Training We applied the default random initialization for All-CNN and used the weights pretrained on ImageNet [5] for both Resnet-18 and MobileNet v2. During training, we minimize the cross entropy loss with stochastic gradient descent (SGD) using an initial learning rate of 0.01, momentum of 0.9, and weight decay of 0.001. We trained all three models over 50 epochs using the same learning rate schedule, i.e. reduce the learning rate to 80% at every 5 epochs.

Caveats It is not exactly a fair comparison in terms of computation cost when All-CNN is trained on 32×32 image sizes and ResNet-18 and MobileNet v2 are trained on 224×224 ones. Thus, we also implemented a modified ResNet architecture referencing [9], which takes 32×32 images as input. We will refer to this model as ResNet-20. Compared to the standard ResNet-18, this modified model has fewer channels and down-samples more slowly (i.e. changing stride from 2 to 1 in some layers). Even though this ResNet-20 model is deeper, it actually have significantly fewer parameters.

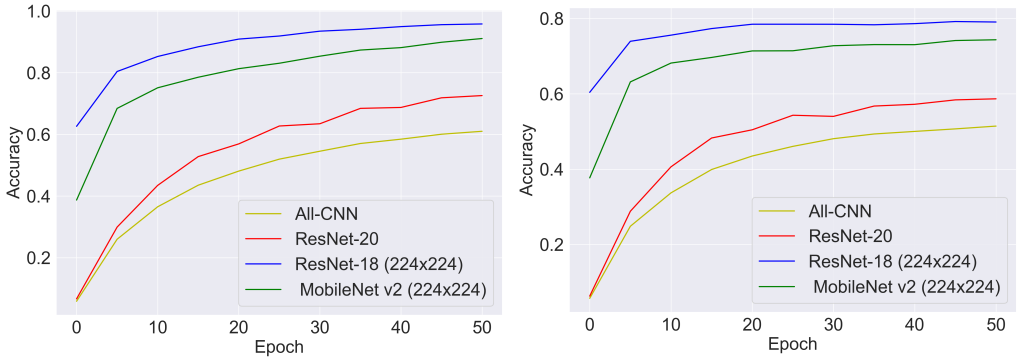
2.3 Computation

2.3.1 Computing Resources

We used a single, personal GPU machine, with an Intel(R) Core (TM) i5-4690K CPU @ 3.50GHz, 32GB system memory. It has a GeForce GTX TITAN X (12GB memory) GPU, with CUDA Version 11.0. See Figure 6 for a screenshot of the machine specs.

```
H/W path          Device      Class          Description
========================================================
/0/63                         processor      Intel(R) Core(TM) i5-4690K CPU @ 3.50GHz
/0/100/1/0                    display        GM200 [GeForce GTX TITAN X]
```

Figure 6: Machine specs.



(a) Training

(b) Testing

Figure 7: Accuracy over epochs.

While we didn't need to use cloud computing services, this machine is very similar to AWS p2.xlarge, which is priced at $0.225/hr under EMR pricing. The computing time we used is less than 20 hours, and thus the cost would have been less than $5.

2.3.2 Environment Set-up

All the code was written in Python. The ML pipeline was primarily implemented in `PyTorch`. In addition, we used basic libraries such as `numpy`, `matplotlib`, `tensorboard`.

2.3.3 Challenges

ResNet-18 and MobileNet v2 pretrained on ImageNet expects images of size 224×224. Upsampling the 32×32 images to 224×224 exceeded the 32GB system memory available. To address that, we preprocessed the images batch by batch during training. We took advantage of that and augmented the training dataset by randomly resizing and cropping the original images. On the flip side, this requires preprocessing the images again at each epoch and significantly increased the training time.

Another related problem is that larger image size results in higher GPU utilization. While we trained All-CNN with a batch size of 500, this exceeds the GPU memory available with both ResNet-18 and MobileNet v2. Thus, we progressively reduced batch size to fit within the 12GB GPU memory. This results in a batch size of 256 for ResNet-18 and a batch size of 100 for MobileNet v2.

2.4 Results

The training and testing accuracy over epochs are summarized in Figure 7. ResNet-18 is the best-performing, closely followed by MobileNet v2. Both of these models are larger and use pretrained ImageNet weights, which explains their superior performance. The ResNet-20 variant, which expects 32×32 images, did not perform as well as its pretrained counterpart. Note that we used a fixed learning rate schedule for all models, and thus the accuracy here does not represent the best achievable accuracy by these models.

In Figure 8, we summarize the three-way relation between accuracy vs. model complexity vs. computation cost (showing GPU utilization at inference here). In the models we considered, the accuracy does increase with model complexity, except for All-CNN, which is an older architecture. In [3], it is pointed out that there is not a linear relationship between model complexity and accuracy, i.e. some models use it parameters more efficiently than others. Using accuracy density as criterion, defined as accuracy normalized by number of parameters [3], ResNet-20 uses its parameters most

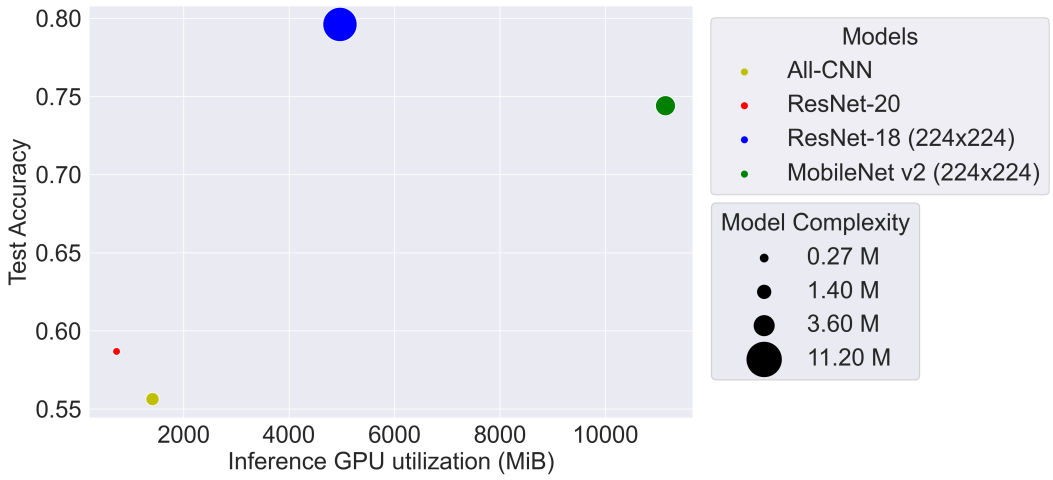


Figure 8: Ball chart reporting accuracy vs. computation cost vs. model complexity.

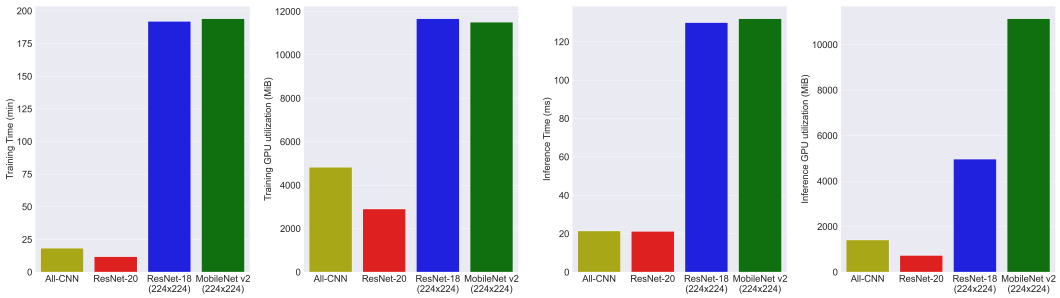


Figure 9: Ball chart reporting accuracy vs. computation cost vs. model complexity.

efficiently among the models considered. Note that ResNet-20 has significantly fewer parameters than ResNet-18 because it uses fewer channels. Another observation in Figure 8 is that there is not a direct relationship between model complexity and GPU memory footprint, evaluated on fixed batch size of 100. Specifically, MobileNet v2 only has a third of the number of parameters of ResNet-18. But, it has more than double the GPU footprint. In fact inference on MobileNet v2 took 11139MiB, which almost depletes the available GPU memory. To understand this counter-intuitive result, we too a closer examination of its architecture. Depthwise separable convolutions technique used by MobileNet v2 reduces the number of parameters, but does not reduce the size of intermediate feature maps. But, MobileNet v2 may be more suited for inference on one image (or very small batch size) at a time on mobile devices, since its model size is relatively small.

Finally, we summarized the computational cost in Figure 9, where we report the GPU utilization and wall-clock time during training and inference. For training, we report the computation cost based on the actual training configurations, e.g. different batch size constrained by available GPU memory. For inference, we report the computational cost over a fixed batch size of 100, averaged over 100 batches. Both the training time and the inference time mostly depends on whether the input images are 32×32 or 224×224. There is surprisingly little variations otherwise. Training ResNet-18 with pretrained weights on 224×224 images took 3hr12min and training ResNet-20 on 32×32 images only took 11.8min. The question boils down to whether 20+% performance improvement is worthwhile for 10x training time.

To summarize, we examine the relationships between performance, model complexity and computational costs on CIFAR100. Even in the small number of models we examined, there appears to be nuances relationship between these metrics.

References

[1] Ifpi issues annual global music report, 2020.

[2] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.

[3] Simone Bianco, Remi Cadene, Luigi Celona, and Paolo Napoletano. Benchmark analysis of representative deep neural network architectures. *IEEE Access*, 6:64270–64277, 2018.

[4] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[6] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[8] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[9] Reproducing cifar10 experiment in the resnet paper, 2020.