Introduction to deep learning

IMAGE CLASSIFICATION USING MNASNET NETWORK

Group 19

Name	**ID**
Lê Ngọc Phan Anh	BI12-010
Hồ Quang Anh	BI12-017
Trần Đức Anh	BI12-018
Trần Ngọc Ánh	BI12-040
Nguyễn Vũ Bách	BI12-044

TABLE OF CONTENTS

I. INTRODUCTION

Mobile devices have become ubiquitous in our daily lives, and the demand for efficient and accurate image classification solutions on these devices is growing rapidly. However, traditional convolutional neural network (CNN) architectures are often too computationally expensive for mobile devices.

MnasNet is a novel neural architecture search (NAS) method that addresses this challenge by simultaneously optimizing for both accuracy and latency. MnasNet achieves state-of-the-art accuracy on ImageNet while maintaining low latency, making it an ideal choice for resource-constrained mobile devices.

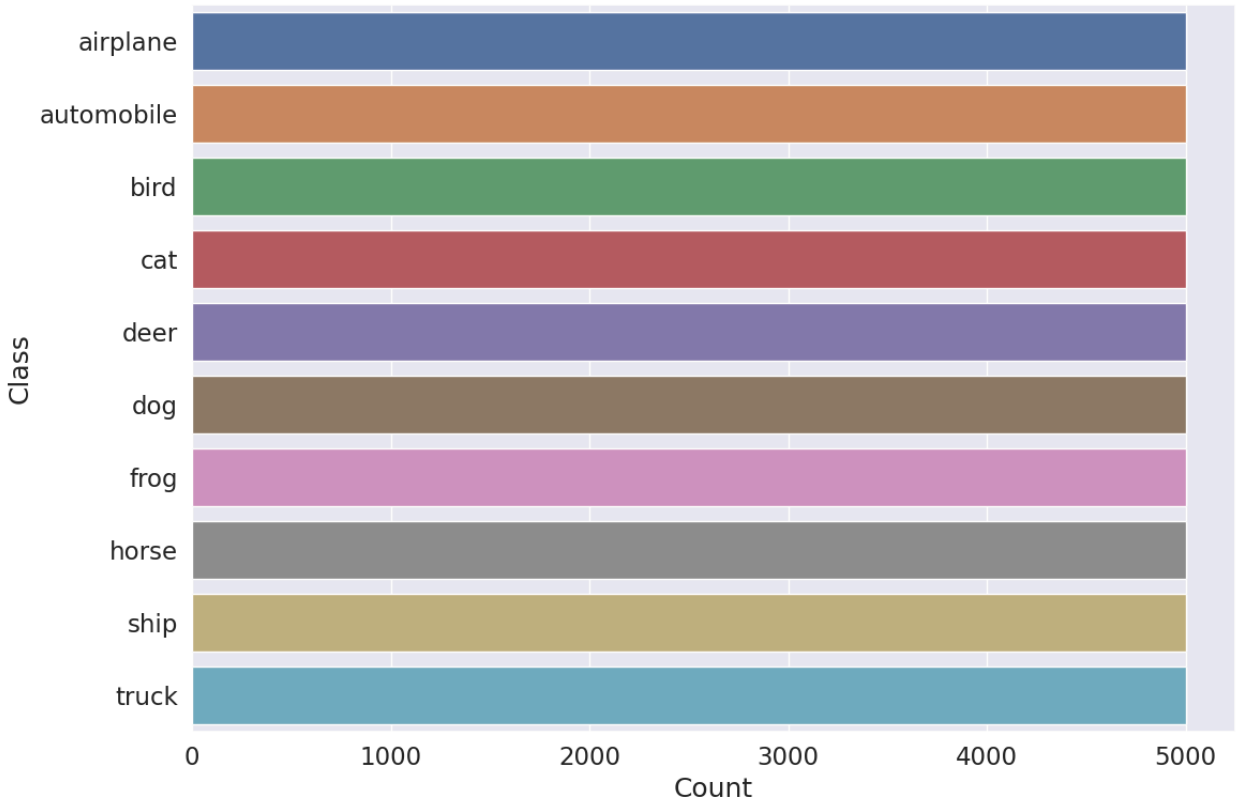
This report delves into the application of MnasNet for image classification, evaluating its performance on various image datasets. We analyze the factors that contribute to MnasNet's performance and discuss its potential for real-time image classification applications.

II. DATASET

The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images, 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some

training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.



Link to download dataset: CIFAR-10

III. MODEL

In this section, we will first discuss our proposed novel factorized hierarchical search space, and then summarize our reinforcement-learning based search algorithm as well as input, out and model architecture.

3.1. Factorized Hierarchical Search Space

Most previous NAS methods search the space to construct a complex cell and then repeatedly stack the same cells with the same configuration to build the whole network; these approaches don't permit layer diversity. However MNasNet defines multiple blocks and searches different hyper-parameters for each block. Therefore we introduce a novel factorized hierarchical search space that factorizes a CNN model into

unique blocks and then searches for the operations and connections per block separately, thus allowing different layer architectures in different blocks. Our intuition is that we need to search for the best operations based on the input and output shapes to obtain better accurate latency trade-offs

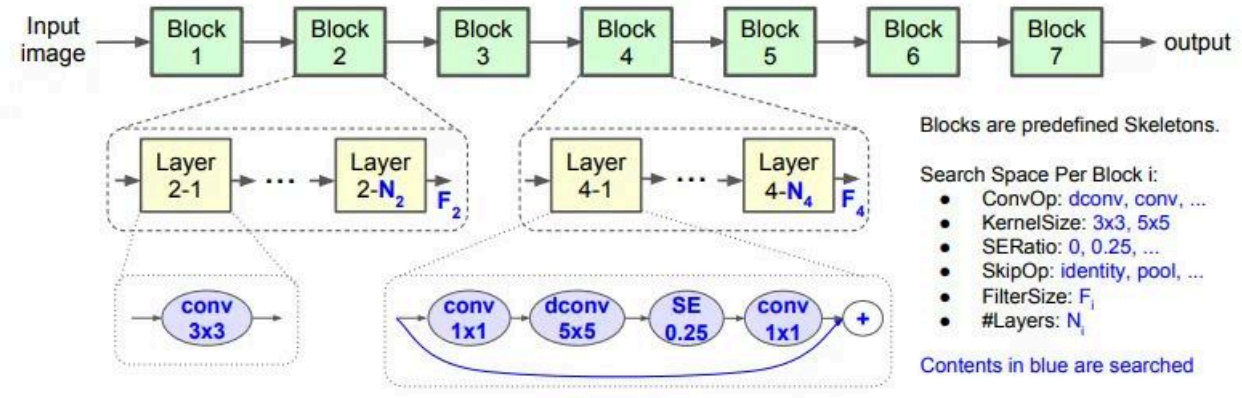


Figure (stt): **Factorized Hierarchical Search Space**.

Network layers are grouped into a number of predefined skeletons, called blocks, based on their input resolutions and filter sizes. Each block contains a variable number of repeated identical layers where only the first layer has stride 2 if input/output resolutions are different but all other layers have stride 1. For each block, we search for the operations and connections for a single layer and the number of layers N, then the same layer is repeated N times (e.g., Layer 4-1 to 4-N4 are the same). Layers from different blocks (e.g, Layer 2-1 and 4-1) can be different.

Here we need to carefully balance the kernel size K and filter size N if the total computation is constrained. For instance, increasing the receptive field with larger kernel size K of a layer must be balanced with reducing either the filter size N at the same layer, or compute from other layers. Figure (stt) shows the baseline structure of our search space. We partition a CNN model into a sequence of pre-defined blocks, gradually reducing input resolutions and increasing filter sizes as is common in many CNN models. Each block has a list of identical layers, whose operations and

connections are determined by a per-block sub search space. Specifically, a sub search space for a block i consists of the following choices:

- Convolutional ops ConvOp: regular conv (conv), depthwise conv (dconv), and mobile inverted bottleneck conv.
- Convolutional kernel size KernelSize: 3x3, 5x5. • Squeeze-and-excitation ratio SERatio: 0, 0.25.
- Skip ops SkipOp: pooling, identity residual, or no skip.
- Output filter size Fi.
- Number of layers per block Ni.

ConvOp, KernelSize, SERatio, SkipOp, Fi determine the architecture of a layer, while Ni determines how many times the layer will be repeated for the block. For example, each layer of block 4 in Figure (stt) has an inverted bottleneck 5x5 convolution and an identity residual skip path, and the same layer is repeated N4 times. We discretize all search options using MobileNetV2 as a reference: For #layers in each block, we search for {0, +1, -1} based on MobileNetV2; for filter size per layer, we search for its relative size in {0.75, 1.0, 1.25} to MobileNetV2.

Our factorized hierarchical search space has a distinct advantage of balancing the diversity of layers and the size of total search space decreased from 10^{39} to 10^{13}.

3.2. Search Algorithm

We use a reinforcement learning approach to find Pareto optimal solutions for our multi-objective search problem. We choose reinforcement learning because it is convenient and the reward is easy to customize

We map each CNN model in the search space to a list of tokens. These tokens are determined by a sequence of actions a1:T from the reinforcement learning agent based

$$J = E_{P(a_{1:T};\theta)}[R(m)]$$

on its parameters θ. Our goal is to maximize the expected reward as where m is a sampled model determined by action a1:T , and R(m) is the objective value:

3.3. Proposed Model Architecture

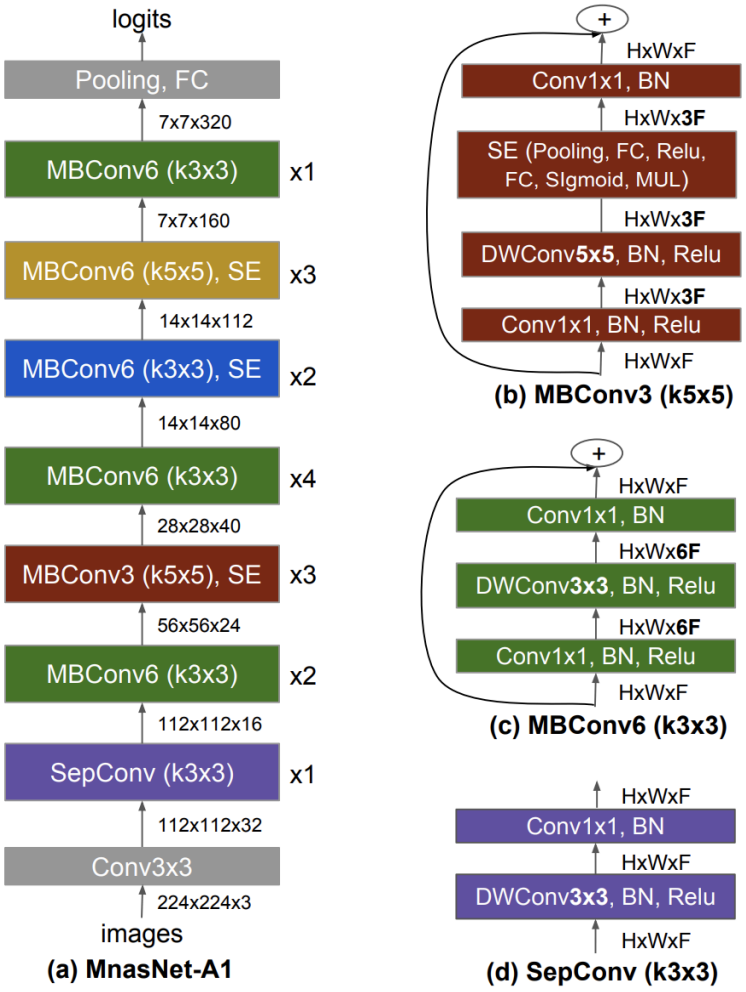


Figure (stt): **MnasNet-A1 Architecture** – (a) is a representative model selected from Table 1; (b) - (d) are a few corresponding layer structures.

MBConv denotes mobile inverted bottleneck conv, DWConv denotes depthwise conv, k3x3/k5x5 denotes kernel size, BN is batch norm, HxWxF denotes tensor shape (height, width, depth), and ×1/2/3/4 denotes the number of repeated layers within the block to be more accurate, there are 31 layers in total and the filter size is 3x3.

To disentangle the impact of our two key contributions: multi-objective reward and new search space, Figure 5 compares their performance. Starting from NASNet, we first employ the same cell-base search space and simply add the latency constraint using our proposed multiple object reward. Results show it generates a much faster model by trading the accuracy to latency. Then, we apply both our multi-objective reward and our new factorized search space, and achieve both higher accuracy and lower latency, suggesting the effectiveness of our search space.

Reward	Search Space	Latency	Top-1 Acc.
Single-obj [36]	Cell-based [36]	183ms	74.0%
Multi-obj	Cell-based [36]	100ms	72.0%
Multi-obj	**MnasNet**	**78ms**	**75.2%**

Table(stt): **Comparison of Decoupled Search Space and Reward Design** – **Multi-obj** denotes our **multi-objective** reward; Single-obj denotes only optimizing accuracy

3.4. Input And Output

In our project The input images are resized to a resolution of 224x224 pixels. The model will predict the class label for each input image from the CIFAR-10 dataset as the output. CIFAR-10 consists of 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

3.5. Model Function

The model takes input images from the CIFAR-10 dataset and passes them through a series of convolutional and pooling layers to extract hierarchical features from the images. The extracted features are then flattened and passed through fully connected layers, which perform classification based on the learned features. The model is trained using optimization algorithms such as stochastic gradient descent (SGD) or Adam, and

the training process aims to minimize a loss function, typically cross-entropy loss, by adjusting the weights of the model. During inference, the trained model takes an input image, performs forward propagation through the network, and outputs class probabilities or predicted class labels for the input image.

IV. IMPLEMENTATION

4.1 Preprocessing dataset

We use torchvision to perform data processing:

- Resize - reduce training time
- RandomHorizontalFlip - generate more training dataset thus creating more powerful and robust image classification
- ToTensor - Convert PIL Image in range [0,255] to torch.FloatTensor in range [0.0 , 1.0]

As we mentioned above, the dataset is divided into five training batches and one test batch, each with 10000 images. To evaluate and fine-tune the model during training, we split the test batch into two parts, 5000 images to test and 5000 others to validate.

4.2 Training model

We use PyTorch Lightning as the framework for training the MnasNet model.

Firstly, we will implement the class LitModel that inherited from lightning.pytorch.LightningModule. The MnasNet model is loaded with pre-trained weights from the ImageNet dataset inside the LitModel.

Secondly, we create a Trainer and utilize the fit() method to start the training of the model with 20 epochs and using a train validation set obtained from the preprocessing part. The training session began.

4.3 Output

Here is the output of the model on the test set which include 5000 images of 10 classes:

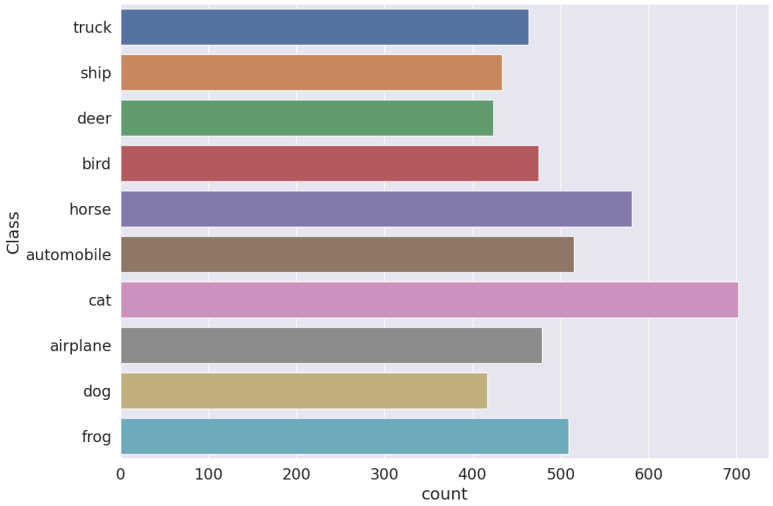


Figure: The distribution of the prediction on the test set.

V. PERFORMANCE

5.1 Accuracy

After having the model trained, we can achieve following figures:

The first one indicates the performance metrics of the model:

Accuracy	88.48%
Precision	89.69%
Recall	88.48%
F1 Score	88.66%

The figure shows that the model is performing well, with high accuracy and a good balance between precision and recall.

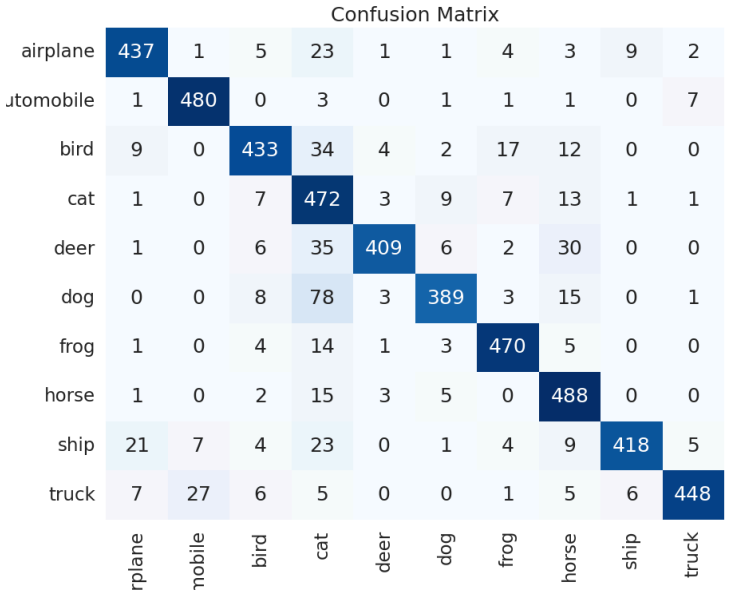


Figure: Confusion matrix of the model

We can observe that the model predicts1 pretty precisely based on the diagonal line of the above matrix.

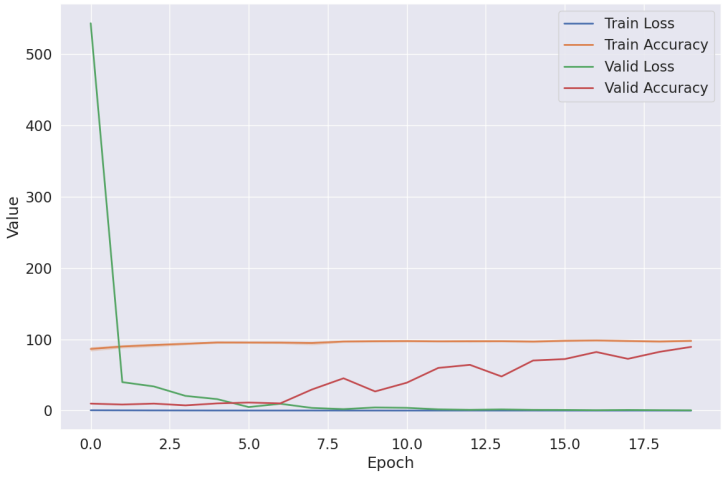


Figure: Changes of the training process

After a certain amount of time training, we can see that the Train Loss has decreased over time and the train accuracy is pretty high since the start of the training because we

have obtained the MnasNet model with pre-trained weights with ImageNet dataset and it fluctuated much over time.

5.2 Latency

Since we do not have the resources and knowledge to run the model on a mobile device to acquire the inference latency, I will cite some information about the latency of the model.

Model	Type	#Params	#Mult-Adds	Top-1 Acc. (%)	Top-5 Acc. (%)	Inference Latency
MobileNetV1 [11]	manual	4.2M	575M	70.6	89.5	113ms
SqueezeNext [5]	manual	3.2M	708M	67.5	88.2	-
ShuffleNet (1.5x) [33]	manual	3.4M	292M	71.5	-	-
ShuffleNet (2x)	manual	5.4M	524M	73.7	-	-
ShuffleNetV2 (1.5x) [24]	manual	-	299M	72.6	-	-
ShuffleNetV2 (2x)	manual	-	597M	75.4	-	-
CondenseNet (G=C=4) [14]	manual	2.9M	274M	71.0	90.0	-
CondenseNet (G=C=8)	manual	4.8M	529M	73.8	91.7	-
MobileNetV2 [29]	manual	3.4M	300M	72.0	91.0	75ms
MobileNetV2 (1.4x)	manual	6.9M	585M	74.7	92.5	143ms
NASNet-A [36]	auto	5.3M	564M	74.0	91.3	183ms
AmoebaNet-A [26]	auto	5.1M	555M	74.5	92.0	190ms
PNASNet [19]	auto	5.1M	588M	74.2	91.9	-
DARTS [21]	auto	4.9M	595M	73.1	91	-
MnasNet-A1	**auto**	**3.9M**	**312M**	**75.2**	**92.5**	**78ms**
MnasNet-A2	**auto**	**4.8M**	**340M**	**75.6**	**92.7**	**84ms**
MnasNet-A3	**auto**	**5.2M**	**403M**	**76.7**	**93.3**	**103ms**

Table 1: **Performance Results on ImageNet Classification** [28]. We compare our MnasNet models with both manually-designed mobile models and other automated approaches – *MnasNet-A1* is our baseline model;*MnasNet-A2* and *MnasNet-A3* are two models (for comparison) with different latency from the same architecture search experiment; *#Params*: number of trainable parameters; *#Mult-Adds*: number of multiply-add operations per image; *Top-1/5 Acc.*: the top-1 or top-5 accuracy on ImageNet validation set; *Inference Latency* is measured on the big CPU core of a Pixel 1 Phone with batch size 1.

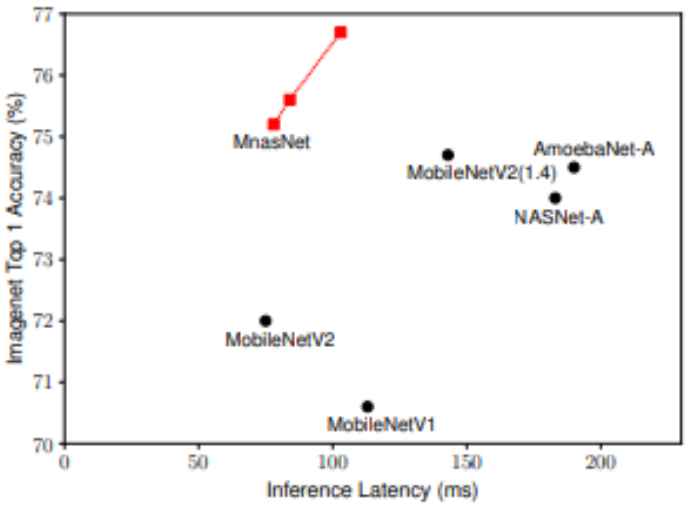


Figure 2: **Accuracy vs. Latency Comparison** – Our MnasNet models significantly outperforms other mobile models [29, 36, 26] on ImageNet. Details can be found in Table 1.

As we can see above, the MnasNet has higher accuracy but with much faster latency compared to some other models such as MobileNetV2, NASNet-A and AmoebaNet-A, etc. That shows why MnasNet is a powerful network in the mobile devices field.

VI. CONCLUSION

In conclusion, the MnasNet model stands as a remarkable achievement in the realm of neural networks, particularly tailored for mobile devices. Its exceptional computational capabilities, combined with meticulous design principles, position it as a powerful solution that adeptly addresses the unique challenges posed by portable computing environments. As we navigate the landscape of technological advancements, MnasNet emerges as a beacon of innovation, offering a harmonious blend of efficiency, adaptability, and cutting-edge functionality. Its impact transcends conventional models, underscoring its significance as a transformative force shaping the future of neural networks within the dynamic sphere of mobile technology.

VII. REFERENCES

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., & Le, Q. V. (2019). MnasNet: Platform-Aware Neural Architecture Search for Mobile. *Conference on Computer Vision and Pattern Recognition (CVPR), 2019*. Retrieved from arXiv / DOI: 10.48550/arXiv.1807.11626