SUSTech Southern University of Science and Technology

Deep Learning Final Project

Pokémon Detection & Recognition

唐云龙 11911607 聂雨荷 11911839 云泽彬 11910912 张通 11911611 郑执 12010126

SUSTech Southern University of Science and Technology

Catalogue

01

Introduction & Dataset & Network

Introduction

SUSTech Southern University of Science and Technology

- Our chosen topic is about object detection
- We have successfully build our own dataset, train the Pokémon detection model and deploy it on Jetson Nano, which can be used to boxed the captured by camera in realtime
- We have used several different network architecture and quantified the model
- We have built an application that take advantage of this model to capturen and introduce Pokémon
- We have deployed a Pokémon fusion model on server to get some interesting generation fusion images of Pokémon

Framework

SUSTech Southern University of Science and Technology

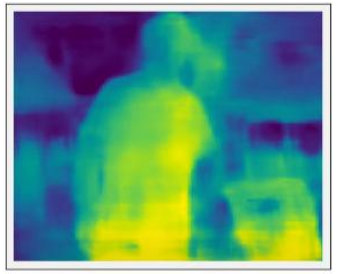
- NVIDIA provide a useful framework for deep learning learners
- Jetson-Inference
- It contains codes (training, apply, label) for model building and deployment and teach you how to build your own model

Image Classification


Object Detection


Semantic Segmentation


Pose Estimation


Mono Depth

https://github.com/dusty-nv/jetson-inference

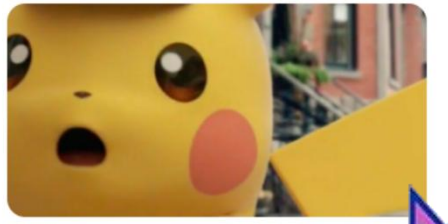
Dataset

SUSTech Southern University of Science and Technology

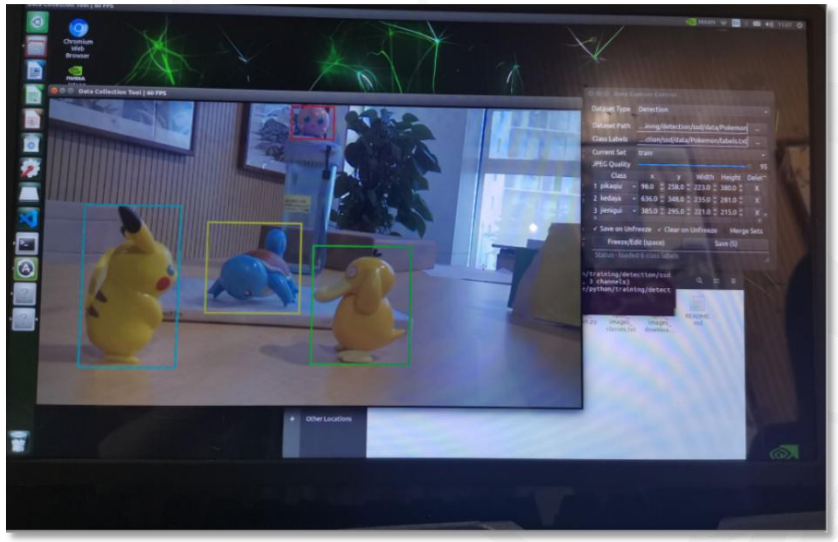
Kaggle Pokémon Dataset

7,000 Labeled Pokemon

7000 hand-cropped and labeled Pokemon images for classification



Handcraft Dataset

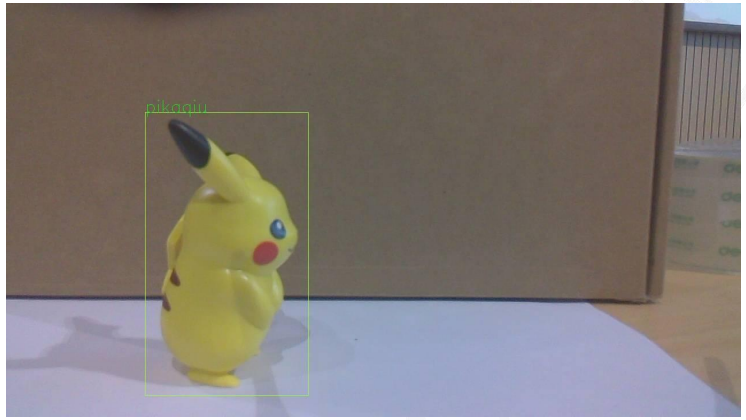


https://www.kaggle.com/datasets/lantian773030/pokemonclassification

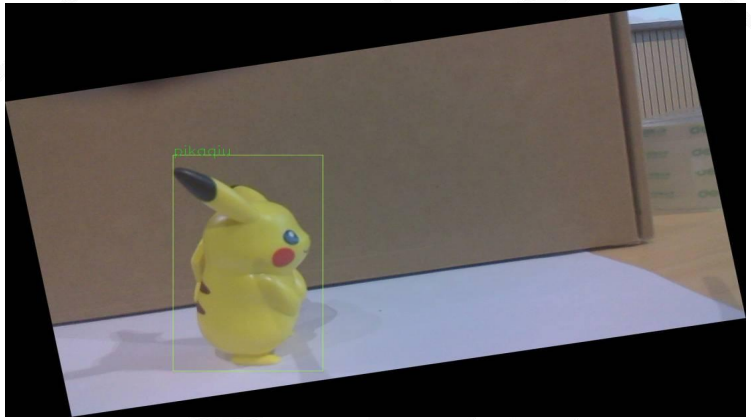
Data Augmentation

SUSTech Southern University of Science and Technology

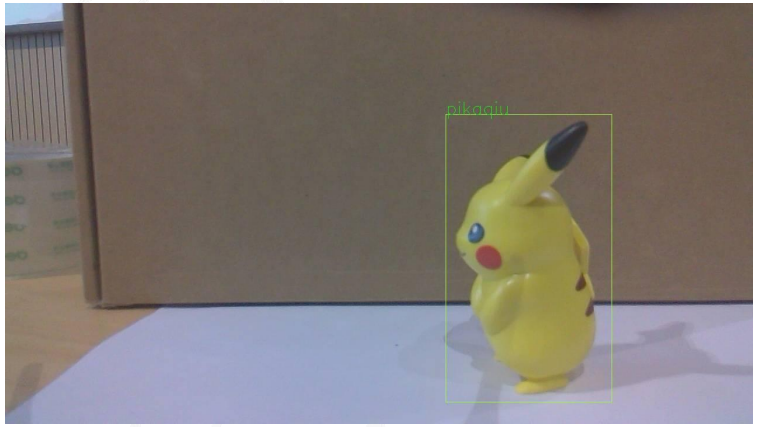
We use several data augmentation methods:



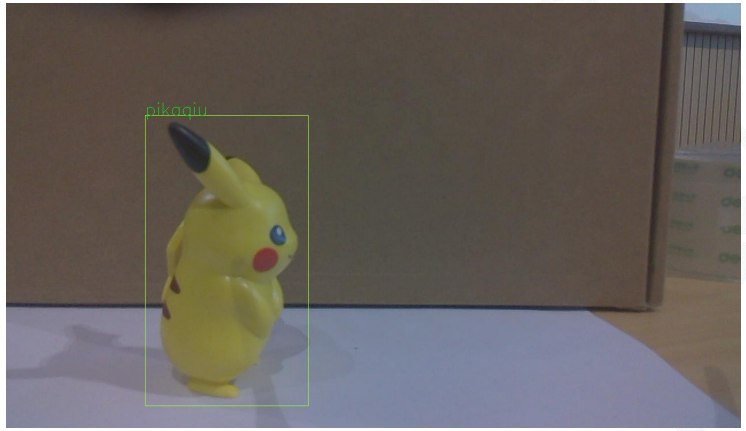
Original



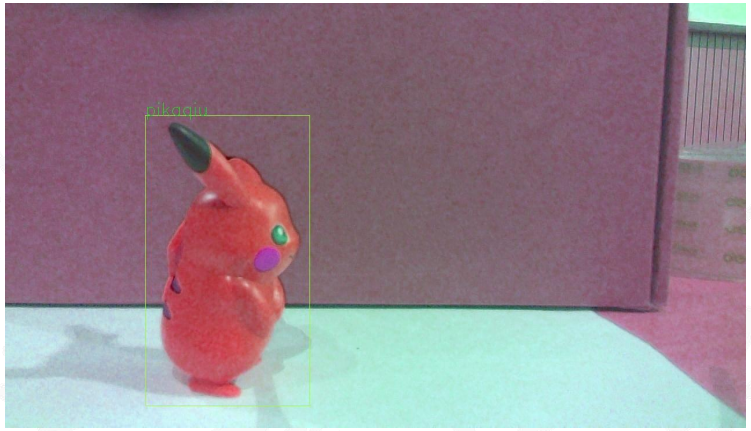
Rotation



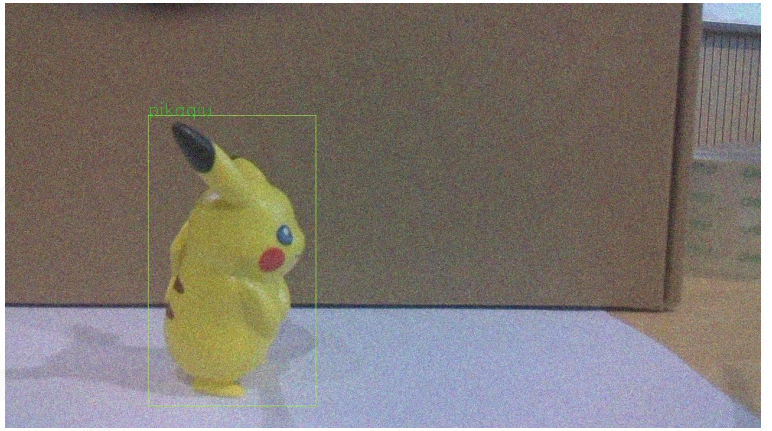
Mirror



Lower Illumination



RGB to HSV

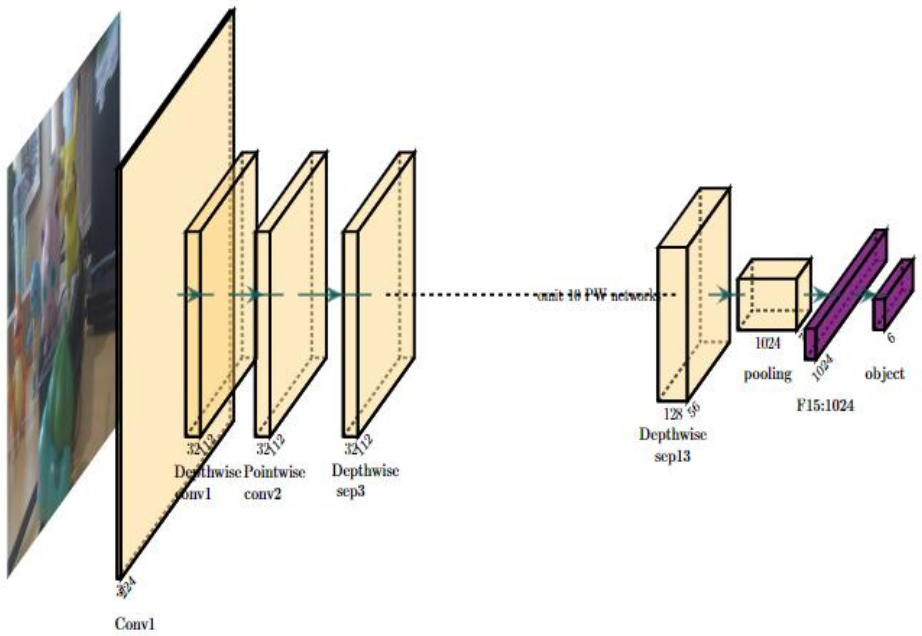


Add Noise

Network Architecture

SUSTech Southern University of Science and Technology

Mobile-V2

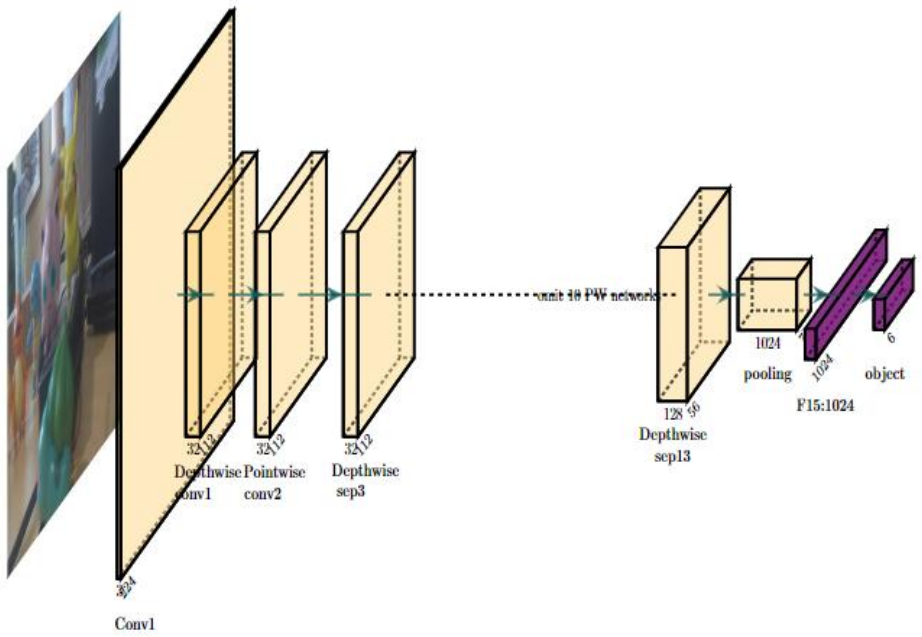


Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	–	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	–	1280	1	1
$7^2 \times 1280$	avgpool 7×7	–	–	1	–
$1 \times 1 \times 1280$	conv2d 1x1	–	k	–	

Network Architecture

SUSTech Southern University of Science and Technology

Mobile-V2



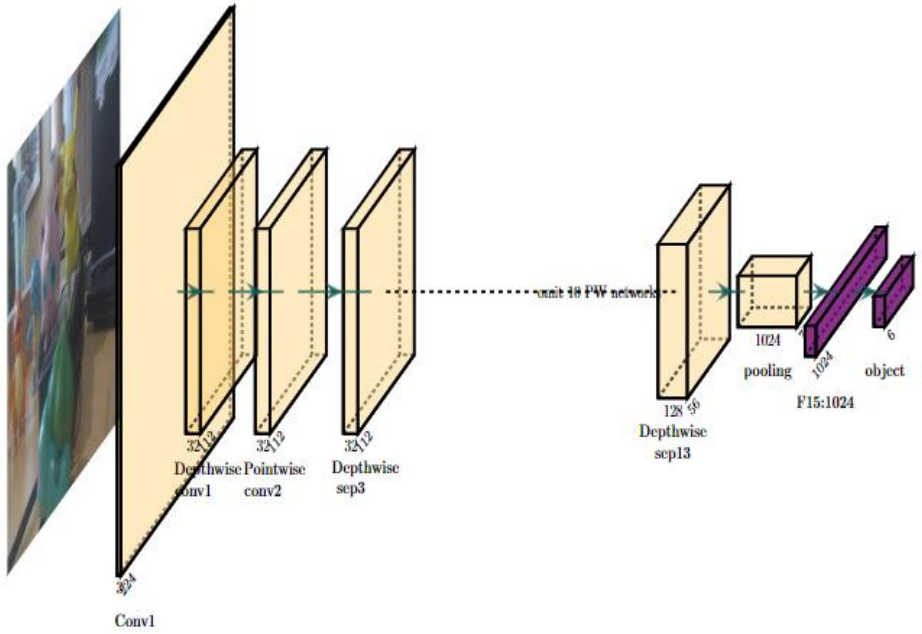

```python
class MobileNetV2(nn.Module):
    def __init__(self, class_num=settings.CLASSES_NUM):
        super().__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 32, 3, stride=2, padding=1, bias=False),
            nn.BatchNorm2d(32),
            nn.ReLU6(inplace=True)
        )
        self.bottleneck1 = self.make_layer(1, 32, 16, 1, 1)
        self.bottleneck2 = self.make_layer(2, 16, 24, 2, 6)
        self.bottleneck3 = self.make_layer(3, 24, 32, 2, 6)
        self.bottleneck4 = self.make_layer(4, 32, 64, 2, 6)
        self.bottleneck5 = self.make_layer(3, 64, 96, 1, 6)
        self.bottleneck6 = self.make_layer(3, 96, 160, 2, 6)
        self.bottleneck7 = self.make_layer(1, 160, 320, 1, 6)
        self.conv2 = nn.Sequential(
            nn.Conv2d(320, 1280, 1, bias=False),
            nn.BatchNorm2d(1280),
            nn.ReLU6(inplace=True)
        )
        self.avgpool = nn.AdaptiveAvgPool2d(1)
        self.conv3 = nn.Conv2d(1280, class_num, 1, bias=False)
```

https://blog.csdn.net/qq_38675397/article/details/104236588?spm=1001.2101.3001.6650.1&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-1-104236588-blog-114368032.pc_relevant_downloadblacklistv1&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-1-104236588-blog-114368032.pc_relevant_downloadblacklistv1&utm_relevant_index=2

Network Architecture

SUSTech Southern University of Science and Technology

Mobile-V2




```
def make_layer(self, repeat, in_channels, out_channels, stride, t):
    layers = []
    layers.append(BottleNeck(in_channels, out_channels, stride, t))
    while repeat-1:
        layers.append(BottleNeck(out_channels, out_channels, 1, t))
        repeat -= 1
    return nn.Sequential(*layers)
def forward(self, x):
    x = self.conv1(x)
    x = self.bottleneck1(x)
    x = self.bottleneck2(x)
    x = self.bottleneck3(x)
    x = self.bottleneck4(x)
    x = self.bottleneck5(x)
    x = self.bottleneck6(x)
    x = self.bottleneck7(x)
    x = self.conv2(x)
    x = self.avgpool(x)
    x = self.conv3(x)
    x = x.flatten(1)
    return x
```

https://blog.csdn.net/qq_38675397/article/details/104236588?spm=1001.2101.3001.6650.1&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-1-104236588-blog-114368032.pc_relevant_downloadblacklistv1&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-1-104236588-blog-114368032.pc_relevant_downloadblacklistv1&utm_relevant_index=2

02

Experiment & Quantization

Experiment

SUSTech Southern University of Science and Technology

Training details

- We trained 400 epochs on a Tesla-V100 GPU and the batch size is 4
- The initial learning rate is 10^{-3} which will reduce 99% every epoch
- The Adam optimizer is adapted and it cost nearly 5 minutes for a single epoch

Ablation validation

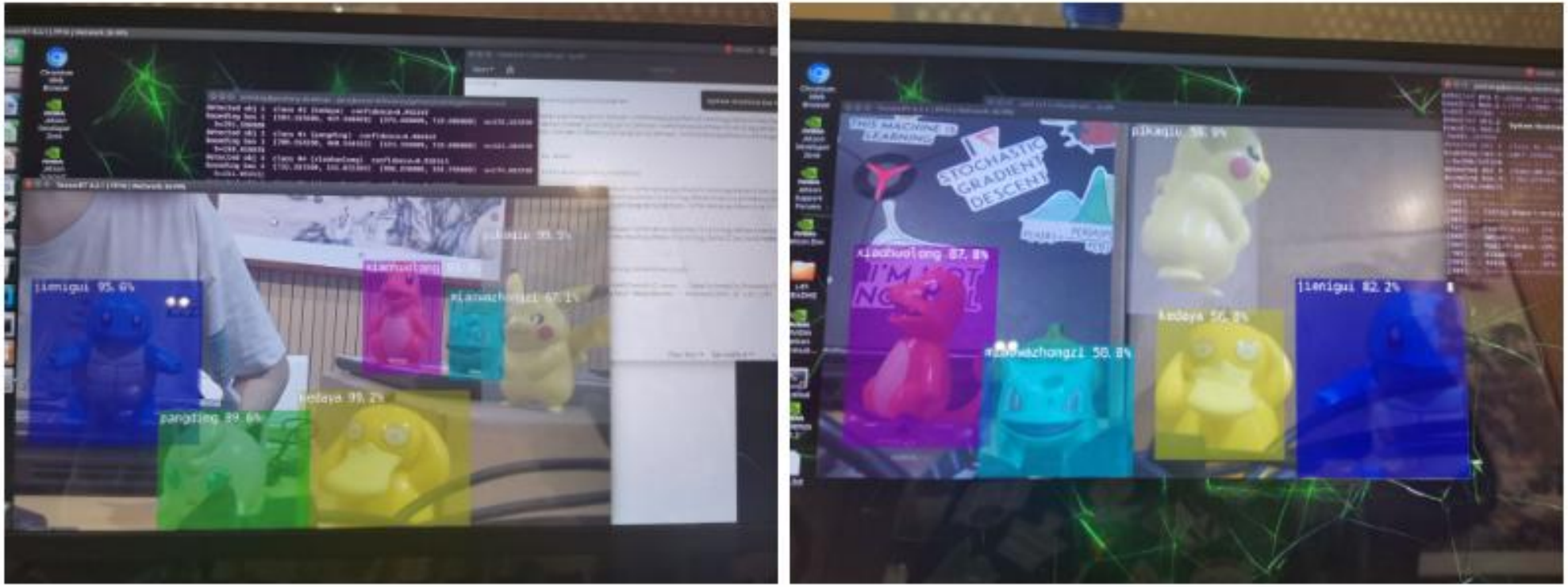
- Our team firstly adopted the MobileNet-v1+ssd as the network but get bad results. We compare both methods by both metrics and examples

(a) Acceptable result (b) Wrong tag (c) Wrong tag (d) Bad bounbing boxes

Experiment

SUSTech Southern University of Science and Technology

Training result



Mobile-V2 Model Training Result

Quantization

SUSTech Southern University of Science and Technology

We use ONNXRuntime to do dynamic and static quantization on the model.However,the jetson-inference does not support quantization operator and we cannot validate our quantized model

Quantization method	Function
dynamic quantization	Quantize the learnable weights of the model
static quantization	Quantize model from a small amount of calibration data
QAT	Minimize model quantization error through finetune training

03

Application & Pokémon Fusion

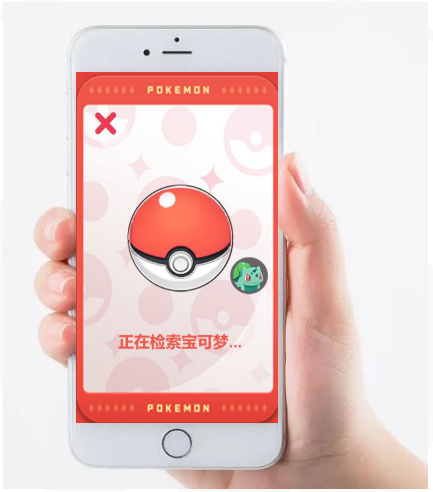
Pokémon-Library

SUSTech Southern University of Science and Technology

We build an application which can introduce the Pokémon captured by the camera

Application Architecture

SUSTech Southern University of Science and Technology



Pokémons Introduction

Pokémons Id: 1,2,3 ...

HTTP getId

Inference

Capture Pokémons

Pokémons Id: 1,2,3 ...

HTTP setId

Cloud Server

Pokémons

Camera

Stream Image

04

Conclusion

Conclusion

SUSTech Southern University of Science and Technology

- Configure and deploy the Jetson Nano environment, including pip, python, c++, jetson-inference project etc.
- Run the base image classification and object detection project, use Jetson nano camera to capture in real time.
- Sample own data set and label the data by ourselves.
- Train our own Pokémon detection model, use different network model and compare the result.
- Build a Pokémon application take advantage of our model based on the real requirement (Support Windows and Android platform).

Conclusion

SUSTech Southern University of Science and Technology

Name	Contribution	Ratio
唐云龙	Model Selection, Training, Adjust parameters, Dataset Construction	20%
聂雨荷	Environment Deployment, Application Development, PPT writing	20%
云泽彬	Dataset Construction, Report writing	20%
张通	Data Augumentation, Quantization, Dataset construction	20%
郑执	Model Experiment, Report writing	20%

THANK YOU!

SUSTech Southern University of Science and Technology

References

- Geissler, D., E. Nguyen, D. Theodorakopoulos, and L. Gatti (2020). Pokérator-unveil your inner pokémon. In ICCC, pp. 500–503.
- Howard, A. G., M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.
- Sandler, M., A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE conference on
- computer vision and pattern recognition, pp. 4510–45