

Group: 29

MobileDet for Object Detection in Remote Sensing Images

Members:

Lưu Bá Anh

Tạ Việt Đức

Phạm Ngọc Huy

1. Abstract

The abstract will provide a brief overview of the research objective, methods, and key findings. It should summarize the role of MobileDet in object detection when applied to remote sensing images, highlighting its efficiency and potential advantages for real-world applications.

2. Introduction

In the modern era of technological advancements, remote sensing has become a vital tool for environmental monitoring, urban planning, agricultural management, and disaster response. Remote sensing images captured by satellites or aerial platforms provide detailed insights into Earth's surface, enabling us to classify and detect various geographical features like forests, agriculture, and urban areas.

However, analyzing these massive volumes of data is complex and requires advanced computational methods. Traditional approaches rely on handcrafted features and are limited in terms of scalability and accuracy. With the advent of deep learning, specifically convolutional neural networks (CNNs), significant improvements in image classification and object detection have been achieved. This report explores how **MobileNetV2**, a lightweight deep learning model, can be effectively applied for object detection in remote sensing images.

3. Problem Statement

The primary problem being addressed in this project is the **automatic detection and classification of geographical features** in remote sensing images. Given a dataset of images representing various geographical categories like forests, river , and industrial areas, the goal is to build a model that can accurately classify these images into their respective categories

1. Varying scales and resolutions of satellite images.
2. Diverse geographical features that might look visually similar but differ in structure.
3. Limited availability of labeled data for training deep learning models.

4. Dataset Used

To evaluate the effectiveness of MobileNet for object detection, we used the **EuroSAT** dataset

The dataset contains a variety of object categories, including:

- AnnualCrop
- Forest
- HerbaceousVegetation
- Highway
- Industrial
- Pasture
- PermanentCrop
- Residential
- River
- SeaLake

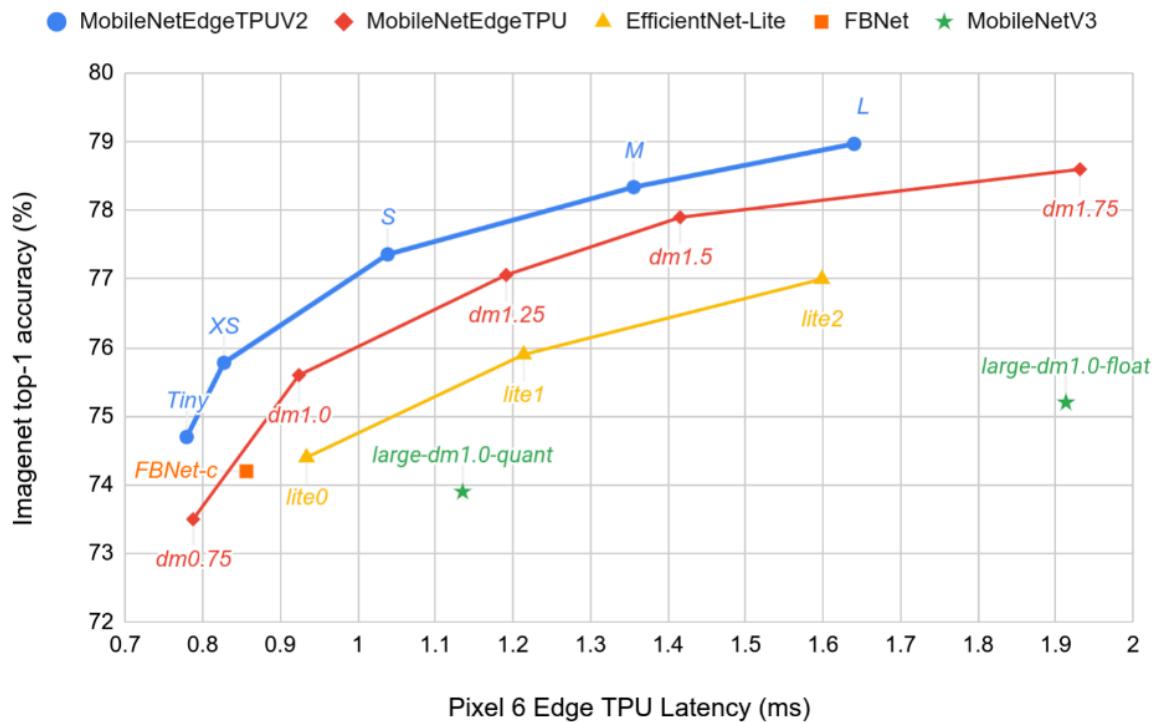
The dataset is organized into **training** and **validation** directories, with each directory containing subfolders for each class. The images in each class are of different geographical areas and conditions, providing the model with a wide variety of visual inputs to learn from.

5. Deep Learning Model: MobileNetV2

MobileNetV2 is a lightweight, efficient deep learning model that balances accuracy and speed. It was designed specifically for mobile and edge devices, making it suitable for remote sensing tasks that require real-time or near real-time predictions. MobileNetV2 employs depth wise separable convolutions, significantly reducing the computational complexity without sacrificing performance.

Key features of MobileNetV2:

1. **Efficient Architecture:** Uses depthwise separable convolutions to reduce parameters and computation.
2. **Pre-trained on ImageNet:** Comes pre-trained on the large-scale ImageNet dataset, which helps transfer learning when applied to remote sensing data.
3. **Low Computational Cost:** Suitable for applications with resource constraints such as edge computing or mobile deployment.



6. Training Process

6.1. Data Augmentation (only available on hardcore training)

To improve the model's generalization ability, the training data is augmented using the **ImageDataGenerator** class in TensorFlow. Various transformations such as random rotations, shifts, shear, zoom, and flips are applied to each image in real-time during training.

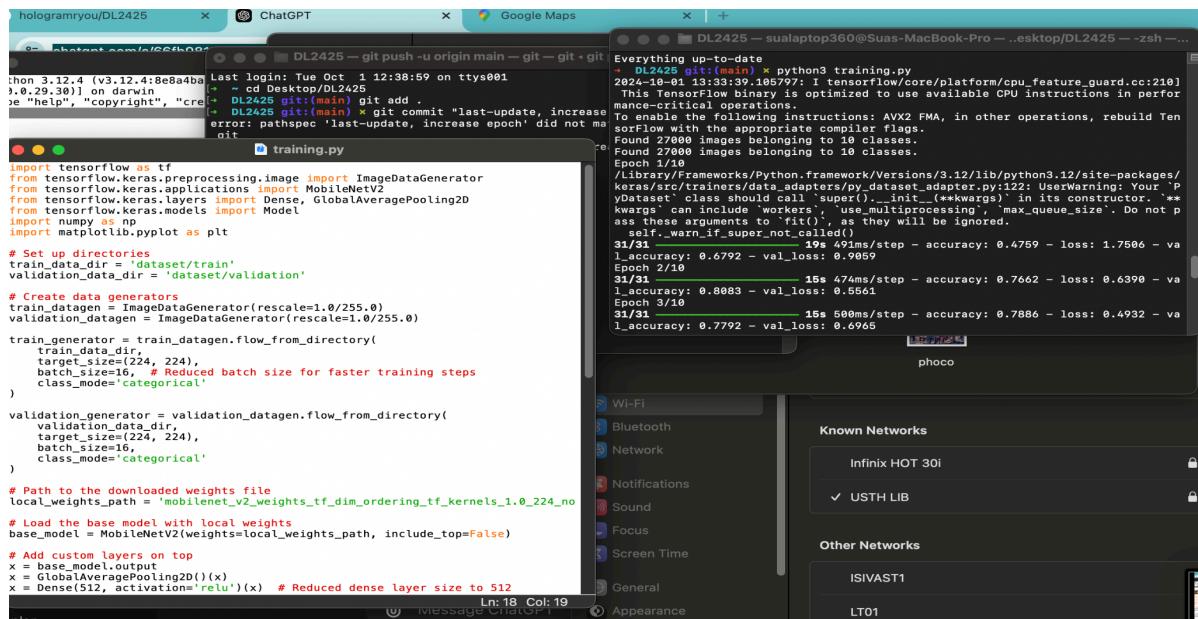
6.2. Training and Validation

The model is trained for 10 epochs with a batch size of 16. During training, both the training and validation accuracies are monitored. The data generators automatically feed images into the model in batches and apply augmentations in real-time.

$$\text{The total steps} = \text{Total samples} / \text{batch size}$$

To reduce the time to process, we can:

- Reduce the Validation Dataset Size
- Increase batch_size



```
hologramryou/DL2425 X ChatGPT X Google Maps X
git:~/Desktop/DL2425$ git push -u origin main --git -git -git -git
Last login: Tue Oct 4 12:38:59 on ttys001
tch 3.12.4 (v3.12.4-8e8a4ba
3.0.29.30) on darwin
↳ "help", "copyright", "cre
↳ DL2425 git:(main) git add .
↳ DL2425 git:(main) git commit "last-update, increase
error: pathspec 'last-update, increase epoch' did not m
oit
Everything up-to-date
DL2425 git:(main) x python3 training.py
2024-10-01 13:33:39.185797: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in perfor
mance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild Ten
sorFlow with the appropriate compiler flags.
Found 27000 images belonging to 10 classes.
Found 27000 images belonging to 10 classes.
Epoch 1/10
1/27000 [00:00 <00:00] 19s 491ms/step - accuracy: 0.4759 - loss: 1.7586 - va
l_accuracy: 0.6792 - val_loss: 0.9059
Epoch 2/10
31/31 [00:00 <00:00] 15s 474ms/step - accuracy: 0.7662 - loss: 0.6390 - va
l_accuracy: 0.8083 - val_loss: 0.5561
Epoch 3/10
31/31 [00:00 <00:00] 15s 500ms/step - accuracy: 0.7886 - loss: 0.4932 - va
l_accuracy: 0.7792 - val_loss: 0.6965
Infinix HOT 30i
USTH LIB
ISIVAST1
LT01
```

import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
import numpy as np
import matplotlib.pyplot as plt

Set up directories
train_data_dir = 'dataset/train'
validation_data_dir = 'dataset/validation'

Create data generators
train_datagen = ImageDataGenerator(rescale=1.0/255.0)
validation_datagen = ImageDataGenerator(rescale=1.0/255.0)

train_generator = train_datagen.flow_from_directory(
 train_data_dir,
 target_size=(224, 224),
 batch_size=16, # Reduced batch size for faster training steps
 class_mode='categorical'
)

validation_generator = validation_datagen.flow_from_directory(
 validation_data_dir,
 target_size=(224, 224),
 batch_size=16,
 class_mode='categorical'
)

Path to the downloaded weights file
local_weights_path = 'mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_224_no_top.h5'

Load the base model with local weights
base_model = MobileNetV2(weights=local_weights_path, include_top=False)

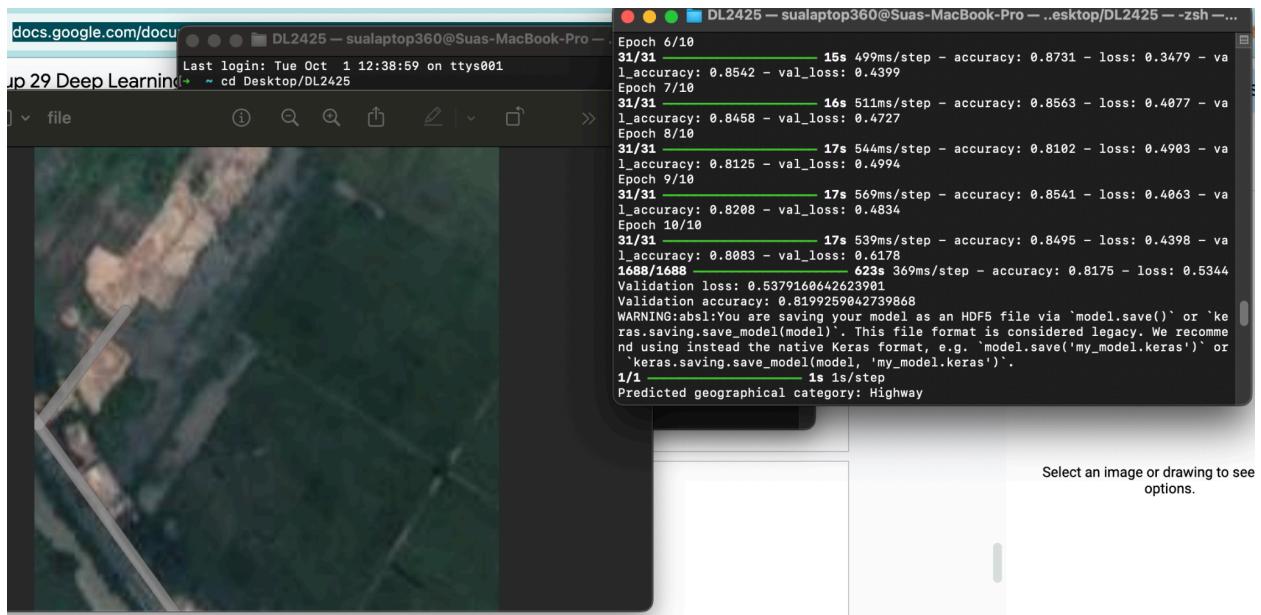
Add custom layers on top
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(512, activation='relu')(x) # Reduced dense layer size to 512

7. Experimental Setup

This section will detail the hardware and software environment used to train and test the MobileDet model. It will include:

- Hardware: Specifications of the machine or platform used (e.g., GPU, CPU, mobile device).
- Software: Frameworks and libraries used (e.g., TensorFlow, PyTorch, OpenCV).
- Training Configuration: Details on training duration, batch size, and any other relevant parameters like model checkpoints.

8. Experiment Results



The screenshot shows a terminal window titled 'DL2425 — sualaptop360@Suas-MacBook-Pro — ..esktop/DL2425 — zsh —'. The terminal displays training logs for a model across 10 epochs. The logs include metrics like accuracy and loss, and a final prediction for a satellite image. The image itself shows a road through a green landscape with a white arrow pointing to it.

```
Epoch 6/10
31/31 15s 499ms/step - accuracy: 0.8731 - loss: 0.3479 - val_accuracy: 0.8542 - val_loss: 0.4399
Epoch 7/10
31/31 16s 511ms/step - accuracy: 0.8563 - loss: 0.4077 - val_accuracy: 0.8458 - val_loss: 0.4727
Epoch 8/10
31/31 17s 544ms/step - accuracy: 0.8102 - loss: 0.4903 - val_accuracy: 0.8125 - val_loss: 0.4994
Epoch 9/10
31/31 17s 569ms/step - accuracy: 0.8541 - loss: 0.4063 - val_accuracy: 0.8208 - val_loss: 0.4834
Epoch 10/10
31/31 17s 539ms/step - accuracy: 0.8495 - loss: 0.4398 - val_accuracy: 0.8883 - val_loss: 0.6178
1688/1688 623s 369ms/step - accuracy: 0.8175 - loss: 0.5344
Validation loss: 0.5379160642623901
Validation accuracy: 0.8199259842739868
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
1/1 1s 1s/step
Predicted geographical category: Highway
```

Select an image or drawing to see options.

8.1. Performance Metrics

After training, the model's performance is evaluated on the validation dataset. The following results were obtained:

- Validation Loss: 0.538
- Validation Accuracy: 0.82

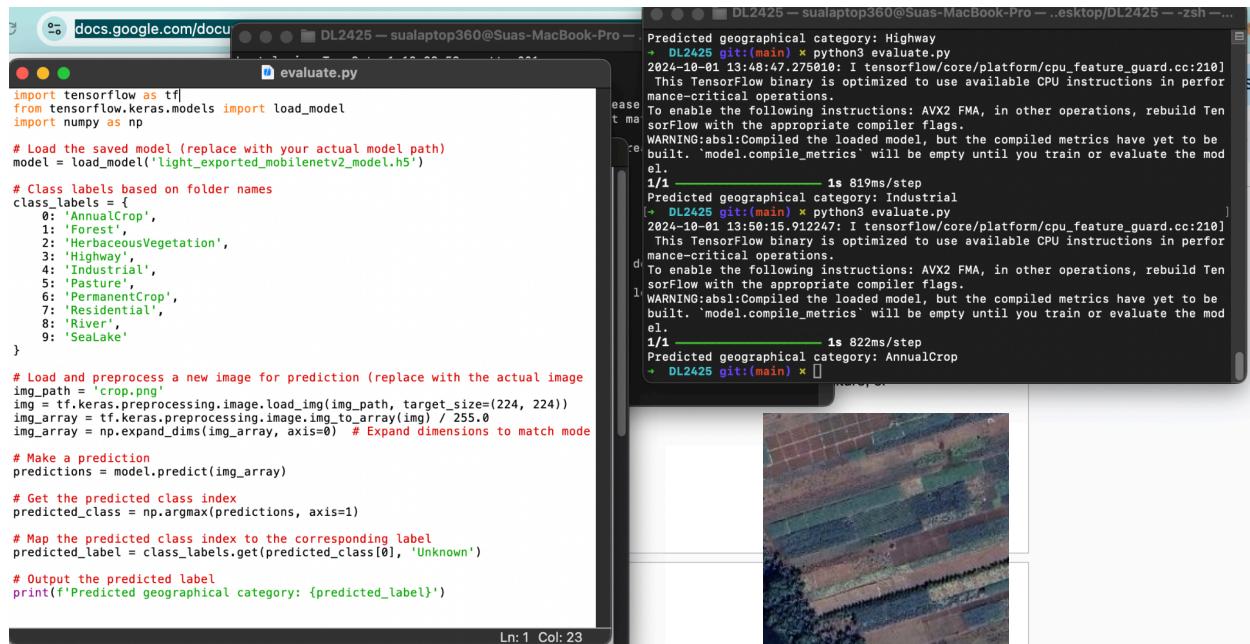
These metrics indicate that the model has learned to generalize well to unseen data.

8.2. Visualization of Training Progress

The training and validation accuracy and loss are plotted over the 10 epochs to visualize the learning curve. The accuracy increases steadily over the epochs, while the loss decreases, showing that the model is converging.

8.3. Prediction on New Data

The trained model is capable of making predictions on new remote sensing images. Given a new satellite image, the model classifies it into one of the categories: forest, agriculture, or urban...



The screenshot shows a terminal window with two panes. The left pane displays a Python script named `evaluate.py`. The right pane shows the command being run and its output. The output indicates the model has been loaded and is ready to predict. It then processes a new image named `'crop.png'` and prints the predicted geographical category as `'AnnualCrop'`.

```
docs.google.com/docs DL2425 - sualaptop360@Suas-MacBook-Pro ~ desktop/DL2425 ~ zsh ~

evaluate.py
Predicted geographical category: Highway
+ DL2425 git:(main) ✘ python3 evaluate.py
2024-10-01 13:48:47.275010: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. 'model.compile_metrics' will be empty until you train or evaluate the model.
1/1 _____ is 819ms/step
Predicted geographical category: Industrial
+ DL2425 git:(main) ✘ python3 evaluate.py
2024-10-01 13:50:15.912247: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. 'model.compile_metrics' will be empty until you train or evaluate the model.
1/1 _____ is 822ms/step
Predicted geographical category: AnnualCrop
+ DL2425 git:(main) ✘
```

Code content from `evaluate.py`:

```
import tensorflow as tf
from tensorflow.keras.models import load_model
import numpy as np

# Load the saved model (replace with your actual model path)
model = load_model('light_exported_mobilenetv2_model.h5')

# Class labels based on folder names
class_labels = {
    0: 'AnnualCrop',
    1: 'Forest',
    2: 'HerbaceousVegetation',
    3: 'Highway',
    4: 'Industrial',
    5: 'Pasture',
    6: 'PermanentCrop',
    7: 'Residential',
    8: 'River',
    9: 'Sealake',
}

# Load and preprocess a new image for prediction (replace with the actual image
img_path = 'crop.png'
img = tf.keras.preprocessing.image.load_img(img_path, target_size=(224, 224))
img_array = tf.keras.preprocessing.image.img_to_array(img) / 255.0
img_array = np.expand_dims(img_array, axis=0) # Expand dimensions to match mode

# Make a prediction
predictions = model.predict(img_array)

# Get the predicted class index
predicted_class = np.argmax(predictions, axis=1)

# Map the predicted class index to the corresponding label
predicted_label = class_labels.get(predicted_class[0], 'Unknown')

# Output the predicted label
print(f'Predicted geographical category: {predicted_label}')
```

9. Discussion

The discussion will analyze the results and offer insights into the strengths and weaknesses of MobileDet in the context of remote sensing. Key points will include:

- Performance Analysis: How well MobileDet handles large, high-resolution remote sensing images compared to other models.
- Challenges: Issues such as small object detection,

- Adaptability: Discussion on the model's ability to adapt to different classes and scales of objects specific to remote sensing, like buildings, vehicles, and natural features.
- Limitations: Possible drawbacks such as reduced accuracy under certain conditions or complexity in fine-tuning for specialized tasks.
- Comparison with State-of-the-Art: A critical comparison with other lightweight and full-scale models in terms of accuracy-efficiency trade-off.

10. Optimization and Future Work

This section will explore potential improvements and optimizations to enhance MobileDet's performance in remote sensing scenarios, including:

- Architectural Modifications: Exploring if modifications like adding atrous convolutions or using spatial pyramid pooling can improve feature extraction for remote sensing.
- Transfer Learning: Using pre-trained models on similar tasks or large-scale datasets to boost initial performance.
- Hardware Acceleration: Investigating the use of specialized hardware (e.g., Tensor Cores, Edge TPUs) for faster inference.
- Future Directions: Suggestions for future research, including multimodal integration (e.g., combining SAR and optical images), and leveraging advanced techniques like attention mechanisms.

11. Conclusion

The conclusion will summarize the study's main findings, emphasizing MobileDet's potential as an efficient solution for object detection in remote sensing. It will reiterate the significance of the work in enabling real-time, resource-constrained applications in fields like disaster management and urban planning. Additionally, it will highlight the importance of ongoing research in refining lightweight models for complex remote sensing tasks.

12. References

Keras 3: Deep Learning for Humans :<https://github.com/keras-team/keras>

EuroSAT dataset: <https://github.com/phelber/EuroSAT>

Model: <https://www.kaggle.com/models/google/mobilenet-edgetpu-v2/tensorFlow2>

