Satellite Image Classification Using deep learning approach

Member: Yuwenqian Chen, Yuqing Wang

1. Introduction

   Satellite image classification serves a critical function across various applications, from land cover mapping and urban planning to environmental monitoring and disaster management. The significant advancements in machine learning and computer vision improve the accuracy and efficiency of satellite image classification. Classification techniques for satellite images can be primarily divided into three categories: automatic, manual and hybrid. Each approach offers unique advantages but also comes with its own set of limitations. Our work focuses on automatic classification to release manual works on classification.

2. Dataset

   2.1. Data Description

   The project utilizes the Dataset-RSI-CB256 for satellite image classification, which is publicly accessible and can be in [4]. Sample images from this dataset are depicted in Figure 1, each sized 256 × 256 pixels. The dataset encompasses four distinct classes obtained from various sensors and Google Maps snapshots: cloudy, desert, green area, and water. With approximately 1,500 images per class, it offers a comprehensive representation.

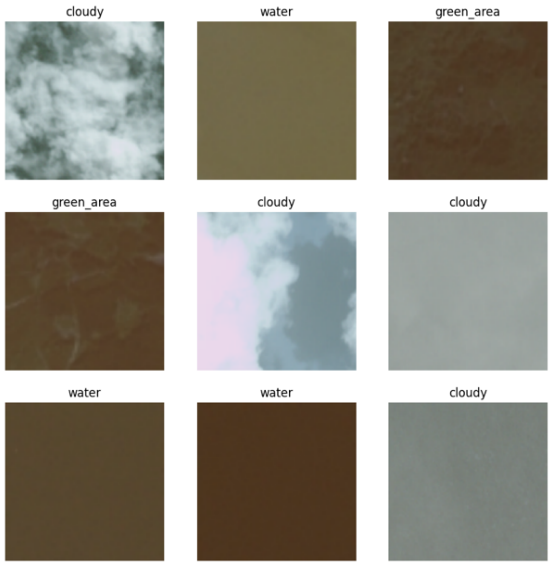


Figure 1: Samples of satellite images from Dataset-RSI-CB256.

   2.2. Data Augmentation

   By expanding the training dataset with data augmentation, this approach enhances the accuracy and generalization capabilities of machine learning models, particularly when data availability is restricted. Through data augmentation, models are exposed to a broader spectrum of training examples, thus enriching their learning experience.

Additionally, this technique mitigates overfitting by introducing variations into the training data, fostering a more resilient and adaptable model.

In implementing data augmentation, we utilized various techniques within the TensorFlow Keras framework to enhance the diversity and robustness of the training dataset. Specifically, we applied tf.keras.layers.RandomFlip to randomly flip images horizontally, tf.keras.layers.RandomRotation(0.2) to introduce random rotations within a range of ±20% of 360 degrees, and tf.keras.layers.RandomContrast(0.2, 1) to adjust the image contrast by a factor between 0.2 and 1, thereby increasing the model's ability to generalize from the training data to new, unseen data.

3. Implementation

3.1. Methodology

This study aims to conduct experiments with Convolutional Neural Networks (CNNs) utilizing pre-trained models through transfer learning, with the objective of enhancing their performance. By employing these techniques effectively, it is anticipated that the proposed approach in this academic investigation will outperform previous studies. The experiment primarily focuses on transfer learning, where specific pre-trained models—MobileNet V2, VGG-16, and Xception—are chosen as the foundational architectures for the main models.

3.1.1. VGG-16

VGG16 is a deep convolutional network for image recognition developed by Visual Geometry Group from Oxford, which was introduced in 2014. It has 16 layers deep and is very uniform, primarily using 3x3 convolutions and max-pooling throughout.

3.1.1.1. Pros

Simplicity: VGG16's architecture is very uniform and simple, making it easy to understand and implement.

Performance: It achieves excellent accuracy on image recognition tasks and is widely used in the computer vision community for feature extraction.

3.1.1.2. Cons

Size and Speed: VGG16 is quite large and cumbersome, leading to high memory and storage consumption. It is also slower to train compared to newer architectures.

3.1.2. MobileNet V2

MobileNet V2 is a lightweight deep neural network architecture designed for mobile and resource-constrained environments. Introduced by Google researchers in 2018, it builds on the ideas of the original MobileNet, utilizing depth-wise separable convolutions to provide an efficient model that can be deployed on mobile devices.

3.1.2.1. Pros

Efficiency: Designed for mobile devices, it is highly efficient in terms of computational resource usage and power consumption.

Speed: MobileNet V2 is fast and suitable for real-time applications on mobile devices.

Size: The model is small, making it easy to deploy where memory is limited.

3.1.2.2. Cons

Accuracy: While very efficient, it generally offers lower accuracy compared to heavier models like VGG16 or Xception when computational constraints are not as tight.

Sensitivity: The depth-wise separable convolutions can sometimes be sensitive to hyperparameter settings and training regimes.

3.1.3. Xception

Xception, which stands for "Extreme Inception," was introduced by Google in 2017. It is an advanced version of Inception, which replaces the standard Inception modules with depth-wise separable convolutions across the entire architecture.

3.1.3.1. Pros

Performance: It significantly outperforms other models like Inception V3 in terms of accuracy and efficiency on large-scale datasets.

Flexibility: Xception applies modifications to the Inception architecture that allow it to have fewer parameters and be more adaptable to different tasks.

Efficiency: While providing high accuracy, it is also relatively efficient in terms of the number of parameters and computational cost.

3.1.3.2. Cons

Complexity: The architecture is more complex than simpler models like VGG16, which can make it harder to implement and fine-tune.

Computational Demand: Though it is more efficient than VGG16, it still requires significant computational power, making it less suitable than MobileNet V2 for mobile environments.

3.2. Pipeline

The experimental program involves several key steps: First, it fetches the input dataset. Then, it builds and trains a model using one of three different pre-trained models as the base. Finally, it evaluates the trained model using a test dataset and generates predictions on sample data. This process is repeated for each of the three pre-trained models to ensure scientifically valid results.

3.2.1. The images are loaded, then rescaled to the size of 224 × 224.

3.2.2. Split the dataset into training set, validation set, and test set in ratio of 7:2:1.

3.2.3. Add the data augmentation layer.

3.2.4. Pre-trained models, named as base models in this study, are loaded into the program from Tensorflow library.

3.2.5. Freeze one-third layers of the base model

3.2.6. Create the global average pooling layers.

3.2.7. Build the Dropout layer.
3.2.8. The prediction layer which will produce outputs of the model is defined.
3.2.9. Build evaluation metrics for: accuracy, loss, precision, recall, F-1 score, ROC AUC
3.2.10. Using early stopping monitoring the model's performance on a validation set and stopping the training process when the performance no longer improves, to prevent overfitting.
3.2.11. Using the training set and the validation set to train the fine-tuning model for 30 epoch with learning rate = 0.0001, optimizer = Adam, and loss = Categorical Cross Entropy.
3.2.12. Using the testing set to determine the success of the trained model.
3.2.13. Evaluate three models.

3.3. Early Stopping

The Early Stopping callback is used to prevent overfitting by halting the training process. In the program, there are 5 parameters, monitored metric, min_delta, patience, verbose, and restore_best_weights. The min_delta parameter sets the minimum change in the monitored metric to be considered as an improvement, which is defined as 0.001. The patience parameter is crucial as it determines the number of epochs to continue training without improvement before stopping; here, it is set to 10 epochs. In this project, three model are setting a different monitored metric. VGG-16 is set to val_loss. MobileNet V2 is set to val_accuracy. And the Xception is set to val_loss.

4. Result

4.1. Learning curve on training set and validation set

4.1.1. VGG-16
4.1.2. MobileNet V2
4.1.3. Xception

4.2. Six metrics evaluation

TP: Number of correctly predicted positive values, true positives
TN: Number of correctly predicted negative values, true negatives
FP: Number of incorrectly predicted as positive, but actually negative values, false positives
FN: Number of incorrectly predicted negative, but actually positive values, false negatives
Accuracy is the most accepted metric for evaluating a model's overall success.

$$Accuracy = \frac{True}{Total}$$

Precision measures the proportion of true positive predictions among all positive predictions made by the classifier.

$$Precision = \frac{TP}{TP + FP}$$

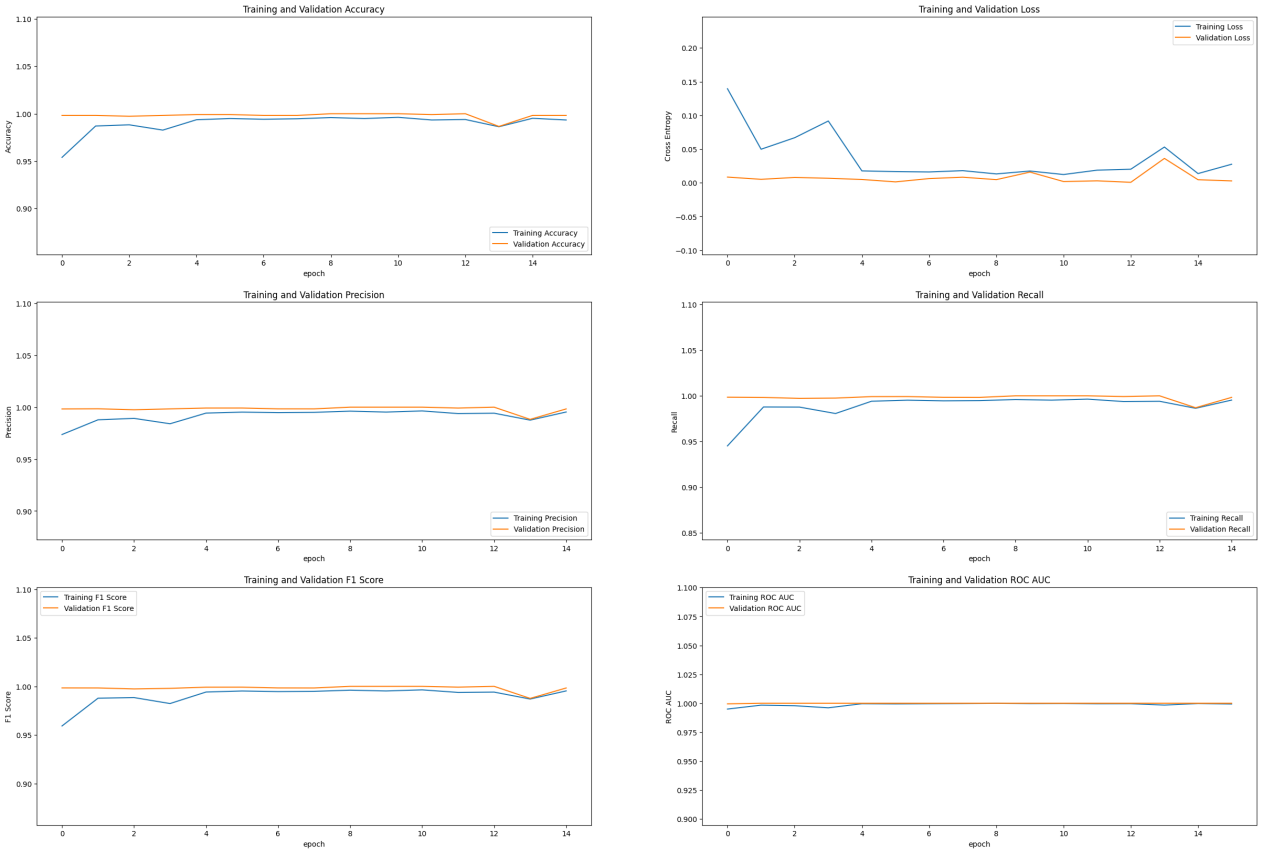
Recall measures the proportion of true positive predictions among all actual positive instances in the dataset.

$$Recall = \frac{TP}{TP + FN}$$

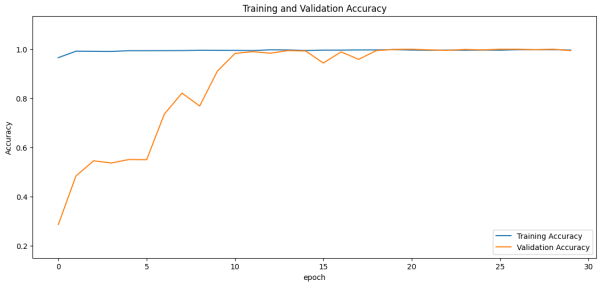
F1 Score is the harmonic mean of precision and recall. It provides a single metric that balances both precision and recall. F1 Score is useful when there is an uneven class distribution or when both false positives and false negatives are important.

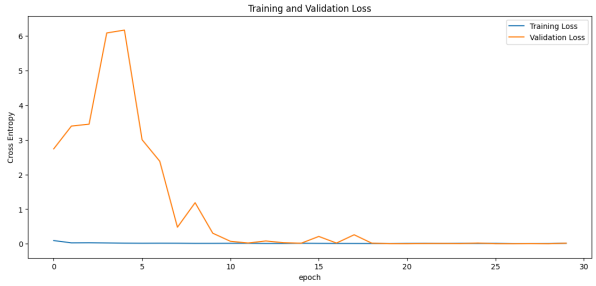
$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

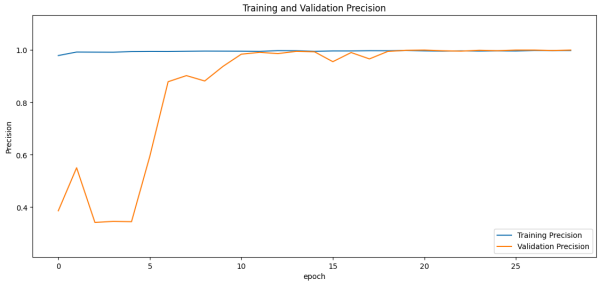
4.2.1. VGG-16

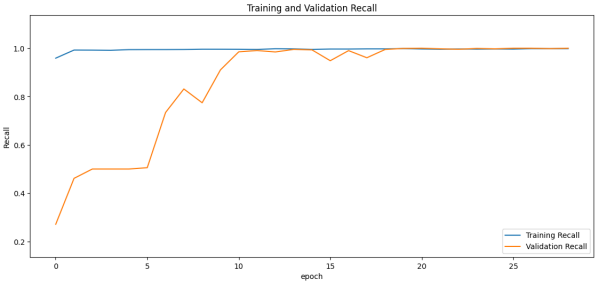


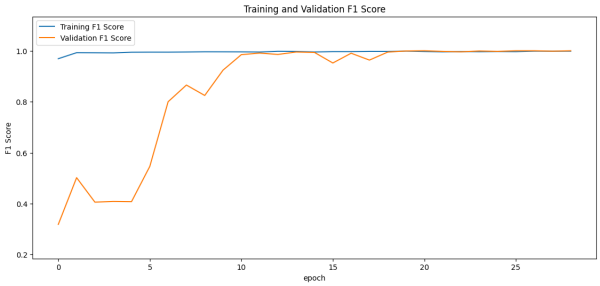
4.2.2. MobileNet V2

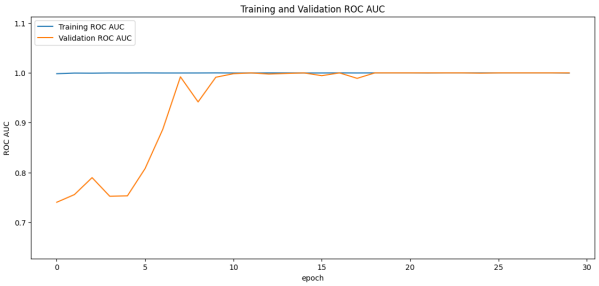




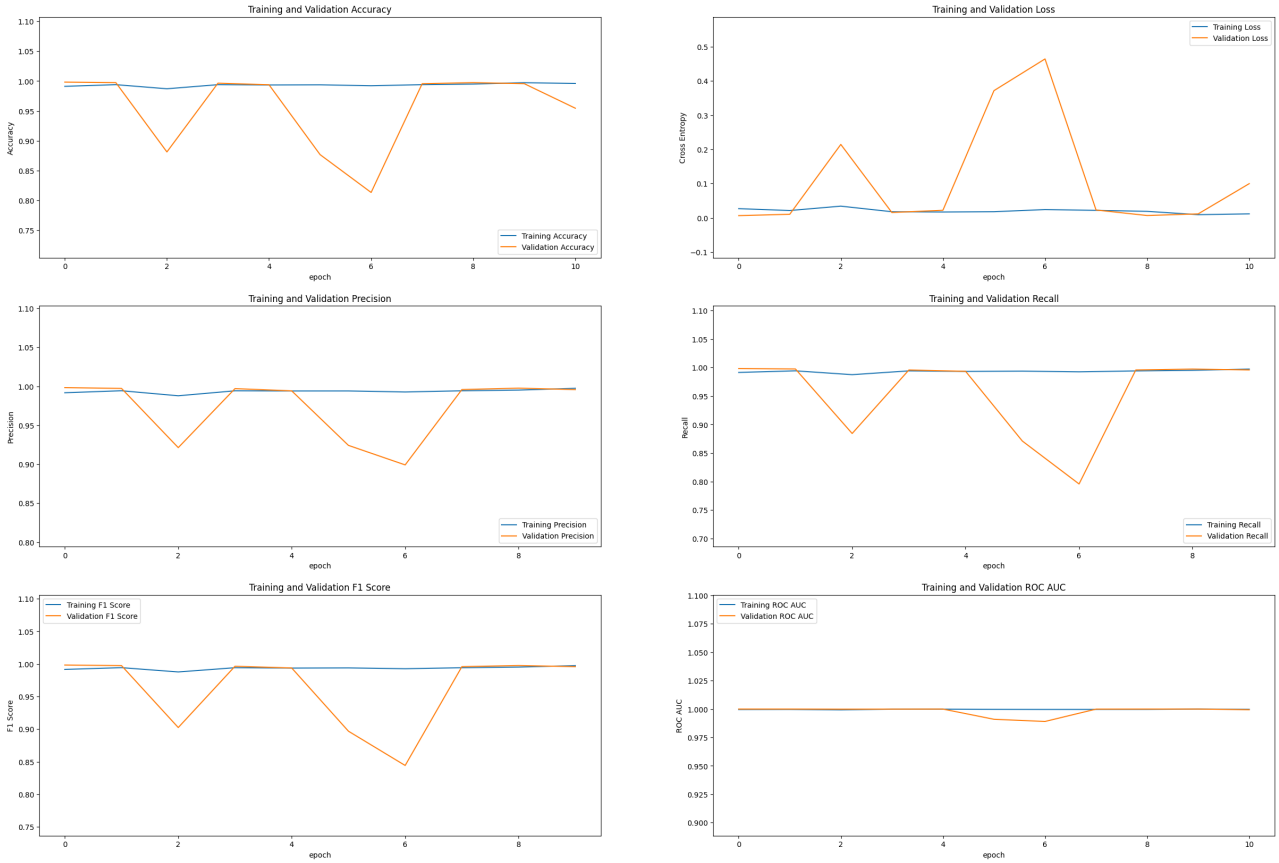








4.2.3. Xception



4.3. The final evaluation on test set

Model	Accuracy	Loss	Precision	Recall	F1 score	ROC AUC
VGG-16	100%	0.00	1.00	1.00	1.00	1.00
MobileNetV2	100%	0.00	1.00	1.00	1.00	1.00
Xception	100%	0.01	1.00	1.00	1.00	1.00

5. Discussion

5.1. Accuracy: All models achieve 100% accuracy on the test dataset, indicating that they correctly classify all instances in the dataset.

5.2. Loss: The loss values are very low (0.00 for VGG-16 and MobileNetV2, 0.01 for Xception), suggesting that the models are effectively minimizing their objective functions during training.

5.3. Precision, Recall, F1 Score, ROC AUC: All models achieve perfect scores of 1.00 for precision, recall, F1 score, and ROC AUC. This indicates that the models make no false positive or false negative predictions and can perfectly distinguish between positive and negative instances.

Based on these results, we can conclude that all three models perform exceptionally well on the given classification task. However, it's important to consider other factors such as the

dataset size, class distribution, and potential biases in the evaluation process. Additionally, it may be beneficial to analyze the models' performance on unseen or real-world data to ensure their generalization ability. Overall, these models demonstrate high effectiveness and reliability in their classification task.

Reference

[1] Tehsin, Samabia, Sumaira Kausar, Amina Jameel, Mamoona Humayun, and Deemah Khalaf Almofarreh. 2023. "Satellite Image Categorization Using Scalable Deep Learning" Applied Sciences 13, no. 8: 5108. https://doi.org/10.3390/app13085108

[2] Guzel, Mehmet et al. “Cloud type classification using deep learning with cloud images.” PeerJ. Computer science vol. 10 e1779. 3 Jan. 2024, doi:10.7717/peerj-cs.1779

[3] Yousaf R, Rehman HZU, Khan K, Khan ZH, Fazil A, Mahmood Z, Qaisar SM, Siddiqui AJ. Satellite Imagery-Based Cloud Classification Using Deep Learning. Remote Sensing. 2023; 15(23):5597. https://doi.org/10.3390/rs15235597

[4] Reda, M. (2021). Satellite Image Classification [Data set]. Kaggle. https://www.kaggle.com/datasets/mahmoudreda55/satellite-image-classification