Satellite Image Classification and Flask web app

https://github.com/sfs-projects/dl

Objective

- Main objective of this analysis is to develop a deep learning model that can classify images into specific classes. The focus of the model is on a specific type of deep learning algorithm, which is a Convolutional Neural Network (CNN).

Description

- Dataset source: Kaggle Satellite Image Classification
- The chosen data set consists of images categorized into four classes: "cloudy," "desert," "green_area," and "water." The images are stored in a dataset directory and have varying sizes and color channels. The goal of this analysis is to train a deep learning model on this data set to accurately classify new images into one of these four classes

Data exploration and processing

- Dataset source: Kaggle Satellite Image Classification
- The dataset contains a total of 5631 images distributed among four classes: 'cloudy', 'desert', 'green_area', and 'water'.
- The number of images per class is as follows:

'cloudy': 1500 images

'desert': 1131 images

'green_area': 1500 images

'water': 1500 images

- The dataset includes images with three different shapes:

(256, 256, 4): Images with an alpha channel

(64, 64, 3): Images without an alpha channel

(256, 256, 3): Images without an alpha channel

```
dataset_dir = "dataset"
target_size = (256, 256)
classes = get_classes(dataset_dir)
print("Classes found in the dataset:", classes, "\n")
analyze_dataset(dataset_dir, classes)
```

```
Classes found in the dataset: ['cloudy', 'desert', 'green_area', 'water']

Number of Images per Class:
cloudy: 1500
desert: 1131
green_area: 1500
water: 1500

Unique Image Shapes:
(256, 256, 4)
(64, 64, 3)
(256, 256, 3)
```

```
train_generator, validation_generator = create_data_generator(dataset_dir, target_size=target_size,
                                                              batch_size=32, validation_split=0.2, augmentation=True)
```

```
Found 4505 images belonging to 4 classes.
Found 1126 images belonging to 4 classes.
```

```
classes = train_generator.class_indices
display_random_images_from_generator(train_generator, classes, num_images=8)
```

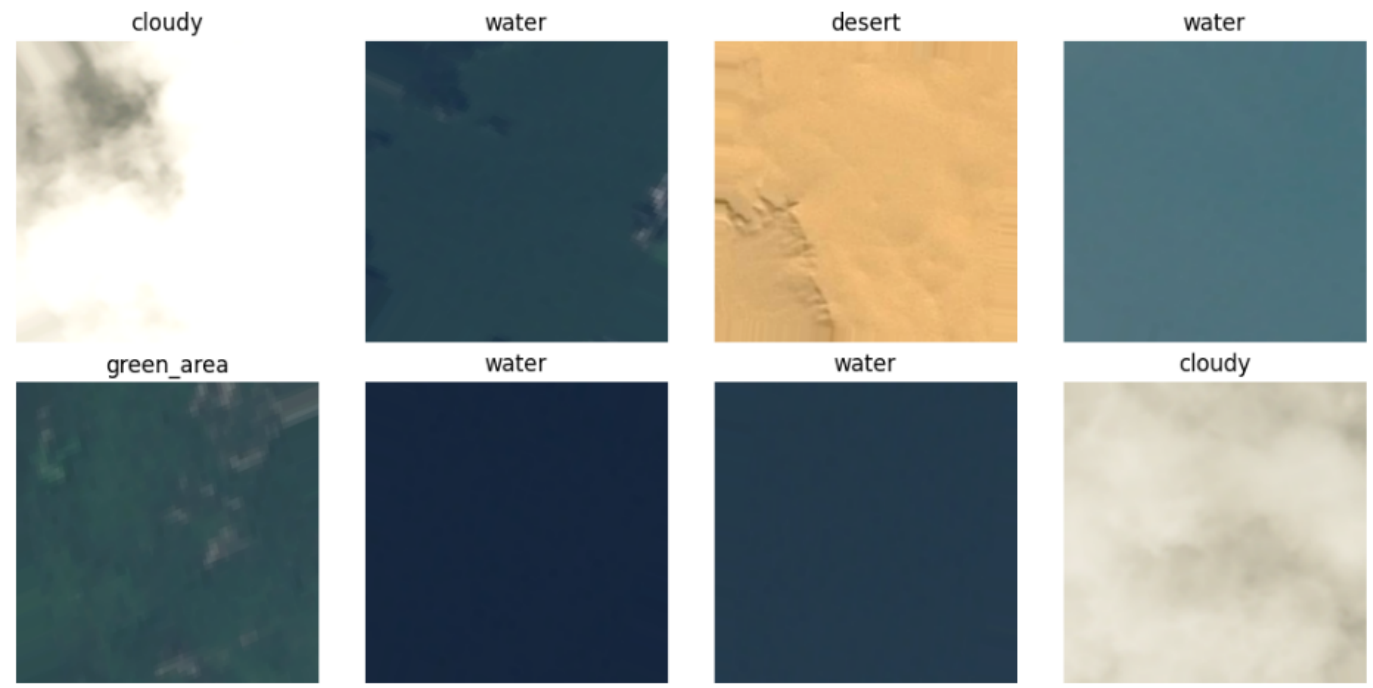


Image preprocessing and data Generators:

- The data generator is created using the ImageDataGenerator class.
- The training set contains 4505 images, and the validation set contains 1126 images.
- Images in the training and validation sets are resized to (256, 256) and are augmented during training.

Model building and training:

- The CNN model architecture consists of convolutional, pooling, dropout, and dense layers.
- The model has a total of 16,839,524 trainable parameters.
- The model is trained for 9 epochs with a batch size of 32 and a learning rate of 0.0001.
- Early stopping is triggered at epoch 9.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 256, 256, 32)      896

 max_pooling2d (MaxPooling2D  (None, 128, 128, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 128, 128, 32)      9248

 max_pooling2d_1 (MaxPooling  (None, 64, 64, 32)       0
 2D)

 dropout (Dropout)           (None, 64, 64, 32)        0

 conv2d_2 (Conv2D)           (None, 64, 64, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 32, 32, 64)       0
 2D)

 dropout_1 (Dropout)         (None, 32, 32, 64)        0

 flatten (Flatten)           (None, 65536)             0

 dense (Dense)               (None, 256)               16777472

 dropout_2 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 128)               32896

 dense_2 (Dense)             (None, 4)                 516

=================================================================
Total params: 16,839,524
Trainable params: 16,839,524
Non-trainable params: 0
```

Model evaluation and visualization

The trained model is evaluated on the validation set, resulting in an accuracy of 85.68%.

The overall metrics for the model are as follows:

- Accuracy: 85.61%
- Precision: 87.52%
- Recall: 85.61%
- F1-Score: 85.31%

Metrics per class:

'cloudy': Precision: 0.98, Recall: 0.85, F1-Score: 0.91

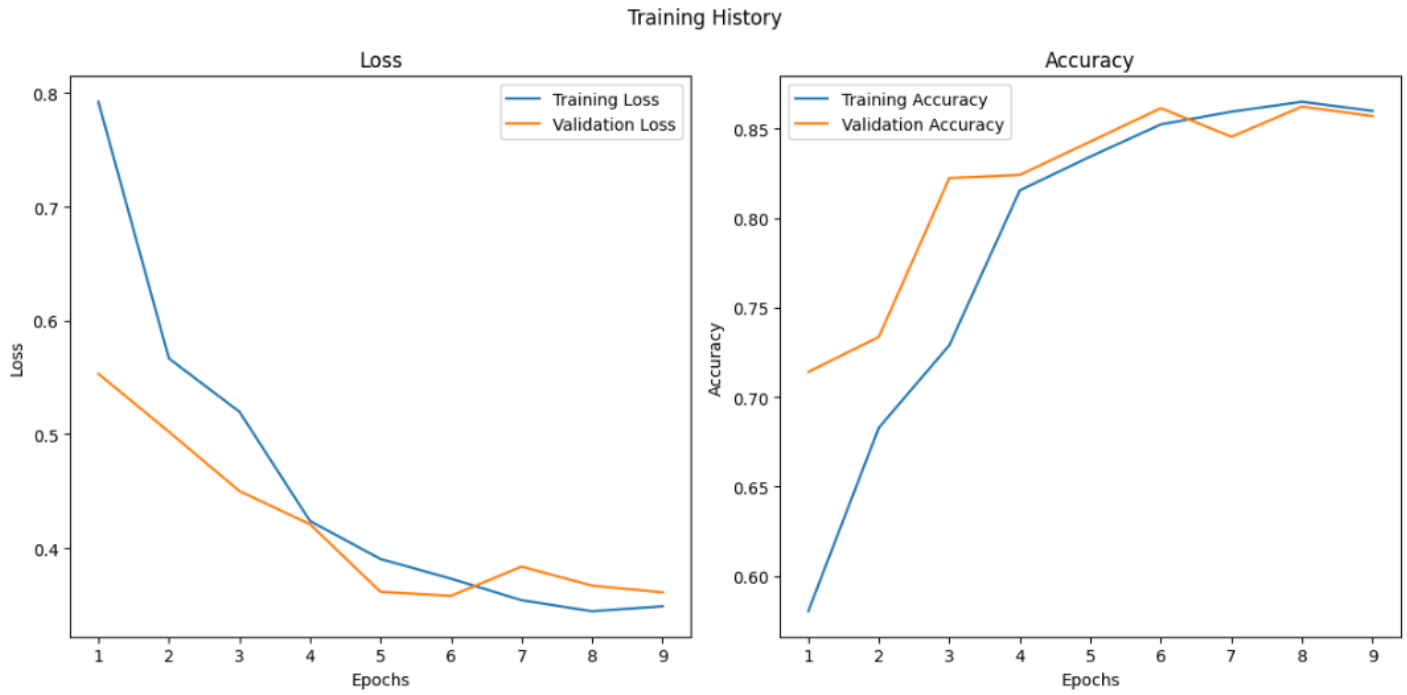
'desert': Precision: 0.86, Recall: 1.00, F1-Score: 0.93

'green_area': Precision: 0.74, Recall: 0.96, F1-Score: 0.83

'water': Precision: 0.92, Recall: 0.65, F1-Score: 0.76

```
saved_model_path = 'best_custom_model.h5'

evaluate_model(saved_model_path, validation_generator)
plot_training_history(history)
```

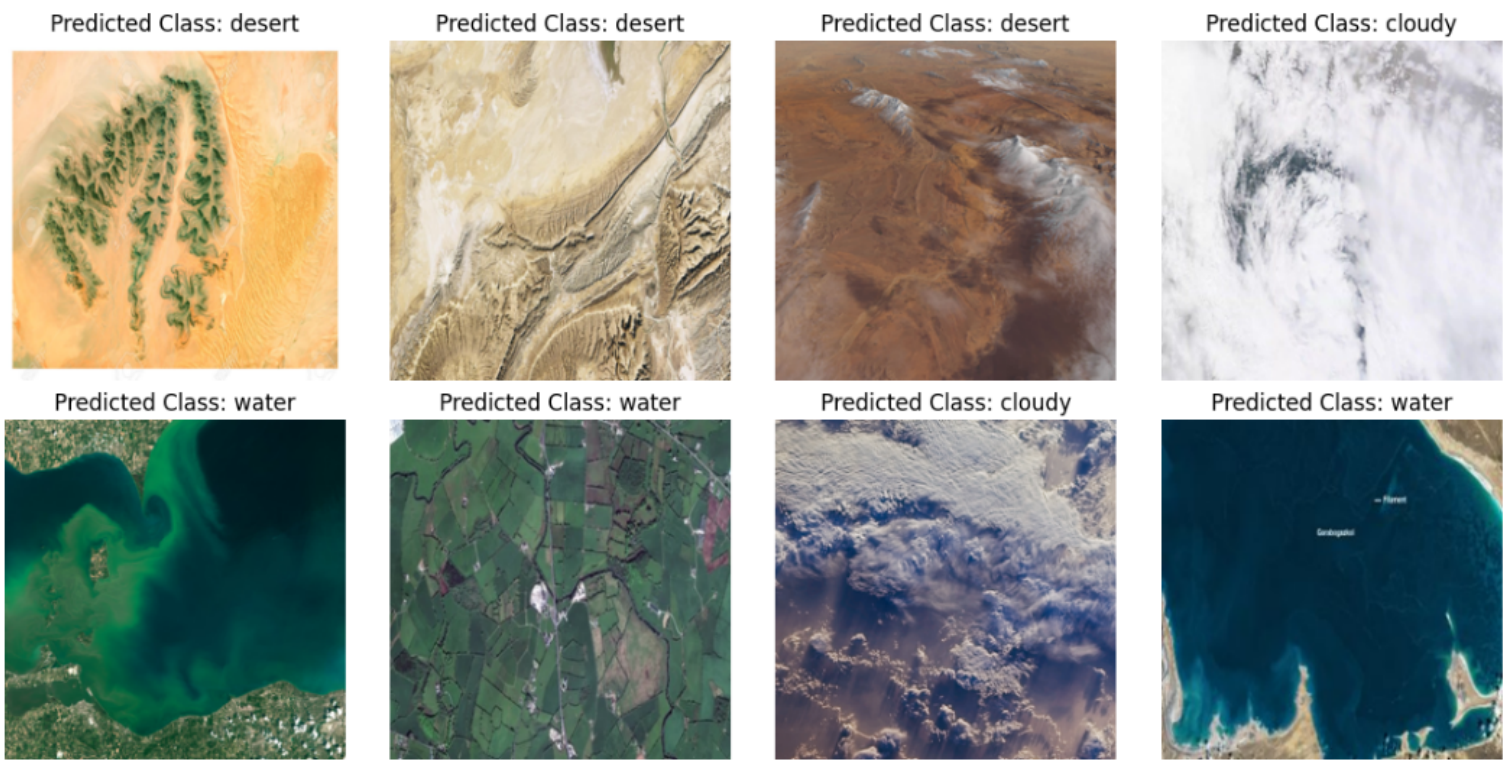
```
36/36 [==============================] - 30s 810ms/step - loss: 0.3546 - accuracy: 0.8597
Test Loss: 0.35464057326316833
Test Accuracy: 0.8596802949905396
36/36 [==============================] - 29s 794ms/step
Overall Metrics:
Accuracy: 0.8561278863232682
Precision: 0.875179706182935
Recall: 0.8561278863232682
F1-Score: 0.8531004531206136

Metrics per Class:
              precision    recall  f1-score   support

      cloudy       0.98      0.85      0.91       300
      desert       0.86      1.00      0.93       226
  green_area       0.74      0.96      0.83       300
       water       0.92      0.65      0.76       300

    accuracy                           0.86      1126
   macro avg       0.87      0.86      0.86      1126
weighted avg       0.88      0.86      0.85      1126
```



Prediction on new data

- The code includes a function to predict classes of new images provided via URLs.
- The model is loaded from a saved file ('best_custom_model.h5').
- The model predicts the classes of the provided images and displays the images along with their predicted class labels.

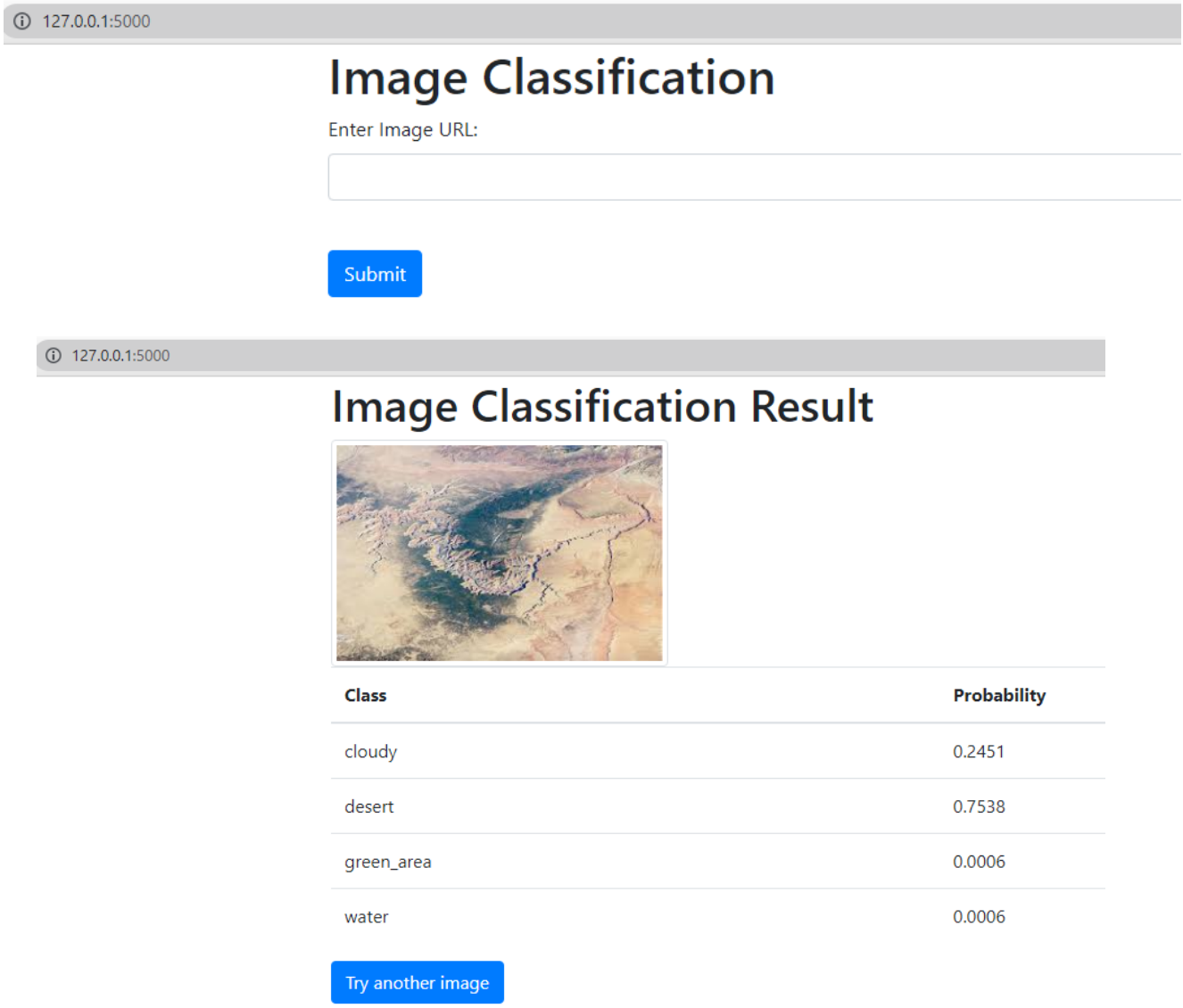
```
1/1 [==============================] - 0s 120ms/step
1/1 [==============================] - 0s 45ms/step
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 44ms/step
1/1 [==============================] - 0s 48ms/step
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 43ms/step
```



Web Application for Image Classification

- To provide an interactive and user-friendly interface for image classification, we developed a Flask web application. The application allows users to input an image URL and receive the predicted class probabilities from our trained model. The following steps outline the functionality of the web application:
- By integrating the Flask web application into our image classification pipeline, we have made the model more accessible to users without requiring them to run code or have programming knowledge. This enhances the practicality and usability of the model, allowing stakeholders to easily classify satellite images for various purposes.

1. Users access the web application through a browser and are presented with a simple interface.
2. The interface includes an input field where users can enter the URL of an image they want to classify.
3. Once the user submits the URL, the application processes the image by fetching it from the provided URL, preprocessing it, and passing it to the trained model for prediction.
4. The model predicts the class probabilities for the image, indicating the likelihood of it belonging to each class.
5. The application displays the image along with a table showing the predicted class labels and their corresponding probabilities.
6. Users can then choose to try another image by clicking the "Try another image" button, which resets the interface for a new prediction.



Key Findings and Insights

- Model achieved good metrics overall and is able to correctly predict most unseen images.
- Even if we try random urls from the internet we can see that 7 out of 8 were classified properly.
- The per-class metrics provide insights into how well the model performs for each category.
- The green areas and water will get some mixed predictions as in the actual dataset we can already see green colors in some water images so its easy to mix them up.

Suggestions

- Increase the size of the dataset: the dataset itself is not the biggest and the quality of the images is quite low.
- Address class imbalance: some classes have fewer images.
- Experiment with larger image sizes, during the preprocessing stage. Increasing the image size may capture more detailed features.
- Try adding more convolutional layers to the model architecture, to potentially capture more complex patterns and enhance the model's feature extraction capabilities.
- Explore alternative model architectures, such as using a pre-trained convolutional neural network (CNN) as the base model, such as VGG16 or ResNet, and fine-tuning it on the satellite image dataset. This transfer learning approach can leverage the pre-trained features and potentially improve accuracy.