The code and pipeline is uploaded to the GitHub repository: <https://github.com/ajaymathew522/CGG_takehome>.

The progress of task can be viewed at <https://github.com/users/ajaymathew522/projects/1>.

The deployed landfill detection application can be accessed using the following link:  
https://landfilldetection-c8382519596b.herokuapp.com/

**RAW DATA**

* The real-world resolution of the images based on the Sentinel-2 program description is 10m. The size of the images is 340.4 MB.
* However, when the area of interests (AOI) are taken from the original images for the purpose of the study, the images have sizes ranging from 65.47 KB to 67.18 KB and shapes ranging from (149, 150, 3) to (155, 156, 3).
* As the AOI images to be used in the task have varying shapes, the shape of the images was downscaled to (128,128,3) so they could be used for model building. The image data was also normalised, so the values were in range of 0 to 1.
* The NDVI, SAVI and MSAVI indexes were also gathered using the pystac api. These images were 1-channel with dimensions ranging from (149, 150) to (155, 156).
* The training data did have any class imbalances as both classes were evenly distributed with 10 images each.

**Modelling**

1. Since this was a binary classification task, the following metrics were used to evaluate the model.

* Accuracy: The proportion of correctly predicted labels.
* Precision: The proportion of true positive predictions among all positive predictions.
* Recall: The proportion of true positive predictions among all actual positives.
* ROC-AUC: The area under the Receiver Operating Characteristic (ROC) curve, which measures the model's ability to distinguish between the two classes.

2. The model achieved an accuracy of 66.67%, precision of 0.6, recall of 1.0, and a ROC-AUC score of 0.8889.

* The model's accuracy is moderate but not high, indicating that it correctly predicts the class in about 66.67% of cases.
* The precision is relatively low (0.6), suggesting that when it predicts the positive class, it has a 60% chance of being correct.
* The recall is high (1.0), meaning that the model captures all the actual positive cases, therefore if the model classifies the image to have landfill, there is a good probability that the site has landfills.
* The ROC-AUC score of 0.8889 is relatively good, indicating that the model can effectively separate the two classes.

3. To improve the model's performance, the following steps would be helpful.

* Hyperparameter Tuning: Experiment with different hyperparameters, such as the number of layers, filter sizes, and learning rates, to find the best configuration for the data.
* Data Augmentation: Augment the training data with rotations, flips, and other transformations to increase the diversity of training examples.
* Regularization: Apply dropout or L2 regularization to prevent overfitting.
* More Data: To provide more training data to enhance the model's ability to generalize. More data can lead to better performance.
* Architectural Changes: Explore more complex architectures or transfer learning using pre-trained models.
* Image stacking: Although NDVI, SAVI and MSAVi images were pulled, these index data could not be integrated into the model. Adding more information to each image would help aid the model learn and classify better.

4. Several factors can influence the performance of a machine learning model, including:

* Data Quality: The quality and cleanliness of your data are paramount. No model can perform well with poor-quality data.
* Data Quantity: Having an insufficient amount of data can lead to overfitting, while having too much data might lead to increased training times.
* Feature Engineering: The choice and engineering of features can significantly impact model performance.
* Hyperparameters: The choice of hyperparameters, such as the learning rate, batch size, and model architecture, can affect the model's performance.
* Regularization: Overfitting can be reduced with the appropriate use of regularization techniques.
* Data Imbalance: Imbalanced class distribution can affect performance, as the model may become biased towards the majority class.
* Model Complexity: A model that is too complex can overfit, while one that is too simple may underfit the data.
* Hardware and Resources: The computational resources available, such as GPU or TPU, can influence the speed of training and choice of model.

**ML Pipeline**

To create a reproducible ML pipeline, the following steps were taken:

* Created ml\_train.py script that reads the training and test data, pulls images from the satellite, trains the model, test the model, and stores the model.
* Created app.py that loads the saved model, reads input from user as geojson files through UI or through json formatted text through the API.
* If geojson file is provided through UI, it checks if the column ‘label’ is present, if so, the script tests the model and displays the accuracy metrics. If the ‘label’ column is not present, the script predicts using the loaded model and then displays the result back to the user.
* Created cgg-model-training workflow that runs the ml\_train.py script whenever a push is made to the repository.
* The deploy\_to\_heroku workflow builds a docker image and containerises the app.py, pushes the image to Heroku and is deployed. The workflow is triggered only if the cgg-model-training workflow is completed.
* GitHub workflows have an easy-to-use UI that could be used to run the pipelines effectively.
* Implementing alerts for performance monitoring and error tracking, as well as setting up automated training schedules with new data, is crucial for monitoring the deployed model.
* Regularly monitoring the model to identify model drift and implementing automated scaling and resource monitoring.
* Tools like Prometheus, Grafana could be utilised for this purpose.