**THEME:**

**AI/ ML based Change Detection for Multi Payload fused Imagery Data**

As a component of challenge 6, it is proposed to develop a AI/ML based analytics on board for EO imagery which can give change detection and also can take intelligent decisions based on the outcome of imagery analysis. The on-board AI system which analyses the data generated from different payloads to glean useful information.

**Ideation:**

**Ideation:** The increasing availability of satellite imagery data presents an opportunity to monitor changes in land cover and land use. However, manual classification of these images is a time-consuming and labour-intensive task. Developing an AI/ML model for automated land-cover classification can enhance the accuracy and efficiency of this process.

**Problem statement:** The aim of this project is to develop an AI/ML model for land-cover classification using satellite imagery data. The objective is to accurately classify different land covers and land use types in the imagery data.

**Benefits:** The model can help in various applications such as urban planning, disaster management, and agricultural monitoring by providing insights into land cover and land use changes over time.

**Tech Stacks used:**

1. Python programming language
2. TensorFlow deep learning framework
3. NumPy and Pandas for data manipulation
4. Scikit-learn for metrics and evaluation
5. Matplotlib for data visualization
6. Google Colaboratory for hosting the notebook and using cloud resources.

**Challenges faced:**

Some of the challenges that can be faced while working on a project related to AI/ML based change detection for multi-payload fused imagery data are:

1. Data collection: Collecting and processing data from different sources can be challenging, as the data can be diverse in terms of format, quality, and resolution.
2. Data preprocessing: Preprocessing of data can be complex, especially when working with multi-payload fused imagery data. This involves aligning the images, removing noise, and normalizing the data.
3. Algorithm selection: Selecting the right algorithms for change detection can be challenging, as there are a large number of available options. Different algorithms may perform better under different circumstances.
4. Training data selection: Selecting the appropriate training data can be difficult, as the data should be diverse enough to cover different scenarios and should be representative of the real-world data.
5. Interpretation of results: Interpreting the results of change detection algorithms can be challenging, as false positives and false negatives can occur. It is important to validate the results and understand the limitations of the algorithm.
6. Computational requirements: Processing and analyzing large amounts of data can be computationally intensive, requiring powerful computing resources and efficient algorithms. This can be a challenge, especially for small teams or individuals with limited resources.
7. Integration of multiple payloads: Integrating data from multiple payloads can be challenging, as each payload may have different characteristics and may require different processing techniques. It is important to ensure that the different datasets are properly aligned and normalized to allow for accurate change detection.

**Dependencies:**

The dependencies for this project are:

1. Python: The programming language commonly used for AI/ML projects
2. NumPy: A Python library for numerical computing
3. Pandas: A Python library for data manipulation and analysis
4. Matplotlib: A Python library for creating visualizations
5. Scikit-learn: A Python library for machine learning tasks
6. TensorFlow or PyTorch: Popular libraries for building deep learning models
7. OpenCV: A library for computer vision tasks
8. Jupyter Notebook: An interactive web-based environment for writing and running code
9. Flask: A web framework for building and deploying web applications
10. Docker: A platform for containerizing applications and their dependencies.

**Risks and Negatives:**

There are a few risks and negatives associated with this project:

1. Dependence on high-quality and up-to-date imagery data: The accuracy and effectiveness of the system depend heavily on the quality and currency of the data used. Outdated or low-quality data can result in inaccurate or misleading results.
2. Overfitting: The system may become overfitted to the specific training data used, resulting in poor performance when applied to new and different data. Regular updates and retraining of the model may be necessary to mitigate this risk.
3. Privacy concerns: The use of satellite imagery data may raise privacy concerns, particularly if it involves the monitoring of individuals or sensitive areas. Appropriate safeguards and ethical considerations must be taken to ensure the responsible use of such data.
4. Complexity and cost: Developing and implementing an AI/ML-based change detection system requires significant technical expertise and resources, including high-performance computing infrastructure and large amounts of training data. This can result in significant costs and complexity in developing and maintaining such a system.
5. Human error: While the use of AI/ML can significantly reduce the manual effort required for change detection, human error in the interpretation and analysis of results can still occur. It is essential to ensure appropriate validation and verification processes are in place to mitigate this risk.

**PROTOTYPE:**

The prototype for this project is a machine learning model that can classify images from the CIFAR-10 dataset into 10 different classes. The model uses a convolutional neural network (CNN) architecture with multiple layers to extract features from the input images and make predictions. The prototype includes the following features:

1. Dataset loading and preparation: The prototype loads the CIFAR-10 dataset and normalizes the pixel values to be between 0 and 1.
2. Model architecture: The prototype defines a CNN architecture with multiple layers, including convolutional layers, max pooling layers, and fully connected layers.
3. Model training and evaluation: The prototype trains the model on the training dataset and evaluates its performance on the test dataset. It also plots the accuracy and loss curves for both the training and validation sets.
4. Hyperparameter tuning: The prototype includes the option to tune hyperparameters such as the learning rate and number of epochs.
5. Future upgrades: The prototype can be further improved by implementing techniques such as data augmentation, transfer learning, and model ensembling to increase its accuracy and robustness.

**Future Upgrades:**

Some potential future upgrades for this project could include:

1. Fine-tuning the model: After training the initial model, we can fine-tune it by tweaking the hyperparameters and experimenting with different architectures to try and achieve better performance.
2. Data augmentation: We can apply data augmentation techniques such as rotation, scaling, and flipping to artificially increase the size of the training set and improve the model's ability to generalize to new images.
3. Transfer learning: We can use a pre-trained model as a starting point and fine-tune it for the CIFAR-10 dataset. This can help us achieve better performance with less training time and fewer resources.
4. Model compression: We can use techniques such as pruning, quantization, and knowledge distillation to reduce the size of the model and make it more efficient for deployment on resource-constrained devices.
5. Ensemble learning: We can combine multiple models trained on different subsets of the data to improve performance and reduce the risk of overfitting.
6. Visualization: We can use techniques such as saliency maps, occlusion analysis, and activation maximization to gain insights into how the model is making predictions and identify areas for improvement.

**DATASET DESCRIPTION:**

CIFAR-10 dataset consists of 60,000 color images in 10 classes, with 6,000 images in each class. The classes include airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The images are of size 32x32 pixels, and each image has RGB channels. The dataset is split into 50,000 training images and 10,000 testing images. The goal of this dataset is to classify each image into one of the 10 classes accurately.

The CIFAR-10 dataset is a well-known dataset that can be used for image classification tasks, and it contains 10 different classes of images. While it may not be specifically tailored for land-cover classification, it can still be used as a starting point for developing an AI/ML model for this task. However, to achieve higher accuracy and relevance, it would be best to use satellite imagery data that is specific to the area of interest and has labelled data for different land cover and land use types.

**IMPLEMENTATION:**

To use the CIFAR-10 dataset for image classification, first, we need to load the dataset using the TensorFlow Keras library. The dataset contains 60,000 32x32 color images in 10 classes, with 6,000 images per class. We then need to pre-process the data by normalizing the pixel values and one-hot encoding the class labels. We can then split the dataset into training and validation sets using a 80:20 ratio. Next, we can define a convolutional neural network (CNN) model using the Keras layers API. The model will have multiple convolutional layers followed by max pooling layers, and then fully connected layers. We can then compile the model using a suitable loss function, optimizer, and metric. Finally, we can train the model on the training data, monitor its performance on the validation data, and then evaluate the model on the test set to obtain the final accuracy score.

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow.keras import datasets, layers, models

# Load the CIFAR-10 dataset

(train\_images, train\_labels), (test\_images, test\_labels) = datasets.cifar10.load\_data()

# Normalize pixel values to be between 0 and 1

train\_images, test\_images = train\_images / 255.0, test\_images / 255.0

# Define the model architecture

model = models.Sequential([

layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)),

layers.MaxPooling2D((2, 2)),

layers.Conv2D(64, (3, 3), activation='relu'),

layers.MaxPooling2D((2, 2)),

layers.Conv2D(64, (3, 3), activation='relu'),

layers.Flatten(),

layers.Dense(64, activation='relu'),

layers.Dense(10)

])

# Compile the model

model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

metrics=['accuracy'])

# Train the model

history = model.fit(train\_images, train\_labels, epochs=10,

validation\_data=(test\_images, test\_labels))

# Evaluate the model on the test dataset

test\_loss, test\_acc = model.evaluate(test\_images, test\_labels, verbose=2)

print("Test accuracy:", test\_acc)

# Plot the accuracy and loss curves for training and validation sets

acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy']

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs\_range = range(10)

plt.figure(figsize=(12, 8))

plt.subplot(2, 2, 1)

plt.plot(epochs\_range, acc, label='Training Accuracy')

plt.plot(epochs\_range, val\_acc, label='Validation Accuracy')

plt.legend(loc='lower right')

plt.title('Training and Validation Accuracy')

plt.subplot(2, 2, 2)

plt.plot(epochs\_range, loss, label='Training Loss')

plt.plot(epochs\_range, val\_loss, label='Validation Loss')

plt.legend(loc='upper right')

plt.title('Training and Validation Loss')

plt.show()