**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 3
2. TensorFlow library for building and training deep learning models
3. NumPy library for numerical computing with Python
4. Matplotlib library for creating visualizations in Python

**Challenges faced:**

common challenges that one might face while working with image classification projects like CIFAR-10 are:

1. Overfitting: This happens when the model becomes too complex and starts memorizing the training data instead of learning the features that generalize well to unseen data. This can be addressed by using regularization techniques like dropout, early stopping, etc.
2. Underfitting: This happens when the model is too simple and fails to capture the relevant features in the data. This can be addressed by increasing the model's capacity, adding more layers or filters, etc.
3. Vanishing gradients: This happens when the gradients during backpropagation become too small to update the model parameters effectively. This can be addressed by using activation functions that don't saturate (e.g., ReLU), using batch normalization, etc.
4. Computational resources: Image classification models like CIFAR-10 can be computationally intensive, especially when training on large datasets or using complex architectures. This can be addressed by using hardware accelerators like GPUs, TPUs, etc.
5. Data augmentation: CIFAR-10 is a relatively small dataset, and overfitting can be a significant challenge. Data augmentation techniques like rotating, flipping, zooming, etc., can be used to artificially increase the size of the dataset and improve the model's generalization performance.

**Dependencies:**

The dependencies for this project are:

1. Python 3.x
2. TensorFlow 2.x
3. NumPy
4. Matplotlib

**Risks and Negatives:**

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

1. Overfitting: There is a risk of overfitting the model to the training data, which may result in poor performance on new, unseen data. To mitigate this risk, techniques such as regularization and data augmentation can be used.
2. Limited dataset: CIFAR-10 is a relatively small dataset, which may limit the performance of the model. Using a larger dataset or combining multiple datasets may improve the model's performance.
3. Computationally intensive: Training deep learning models can be computationally intensive, and may require powerful hardware such as GPUs. This may limit the accessibility of the project to those without access to such hardware.
4. Ethical concerns: There may be ethical concerns related to the use of deep learning models for image recognition, particularly in cases where the models are used to make decisions that affect individuals, such as in facial recognition or surveillance systems.
5. Bias: There is a risk of bias in the model due to factors such as imbalanced data, the selection of features, or the choice of model architecture. To mitigate this risk, it is important to carefully select and pre-process the data, and to thoroughly evaluate the model's performance on diverse datasets.

**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()