# DL_assignment2

Noura Qassrawi

May 17, 2025

## 1 Question 1

Note: in order to run 20 epochs and acheive needed experiments, I had to run the code on my laptop locally while also running some expirements on collab. But the laptop was faster. in some expirements it would have made sense to increase epochs number but its not possible because of time limitation. The objective of this assignment question 1 is to build a land cover classification model using satellite images from the Sentinel-2 satellite. The classification problem involves 9 classes and uses 24,000 images (19,200 for training and 4,800 for validation). As for Question 3, I upload files stored from question 2 for the ensemble, just a disclaimer in case an error pops up that the previous question has to fully run ( Most of it ran and was stored on my local machine as memmory runs out quickly on google collab)

The approach taken involves using a pre-trained VGG16 CNN model as a feature extractor. The extracted features are then used as input to four machine learning models: Logistic Regression, Random Forest, Support Vector Machine (SVM), and XGBoost. The primary goal is to determine which model achieves the best validation accuracy.

### 1.1 Data Loading and Preprocessing

The data is loaded from the provided HDF5 file (`earth_data.h5`). The training data consists of 19,200 images ($64 \times 64 \times 3$), while the validation data consists of 4,800 images ($64 \times 64 \times 3$). Since the images are not normalized, pixel values are scaled to the [0,1] range.

Since Normalization is essential when working with deep learning models, especially convolutional neural networks (CNNs). To ensure data is in a consistent range,images were normalized by dividing each pixel value by 255, as pixel intensities in RGB images range from 0 to 255. This transformation converts the pixel values to the [0,1] range, which is commonly used in deep learning tasks to improve convergence and model performance.

## 1.2 Feature Extraction

The VGG16 model pre-trained on the ImageNet dataset is used for feature extraction. The fully connected layers were removed to use VGG16 as a feature extractor. The output from the CNN is a flattened feature vector of size 2048 for each image.

## 1.3 Model Implementation

### 1.3.1 Logistic Regression

Chosen as a simple and efficient baseline for multi-class classification. Trained on flattened features using the scikit-learn LogisticRegression model. Achieved a validation accuracy of 87.65

Detailed results below:

**Accuracy:** 0.8740
**Precision:** 0.8750
**Recall:** 0.8740
**F1 Score:** 0.8742

**Classification Report:**

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 0.89 | 0.87 | 0.88 | 600 |
| 1 | 0.95 | 0.96 | 0.95 | 600 |
| 2 | 0.89 | 0.87 | 0.88 | 600 |
| 3 | 0.76 | 0.81 | 0.78 | 500 |
| 4 | 0.94 | 0.90 | 0.92 | 500 |
| 5 | 0.83 | 0.87 | 0.85 | 400 |
| 6 | 0.80 | 0.82 | 0.81 | 500 |
| 7 | 0.93 | 0.94 | 0.94 | 600 |
| 8 | 0.84 | 0.80 | 0.82 | 500 |
| Accuracy | | 0.87 (4800) | | |
| Macro Avg | 0.87 | 0.87 | 0.87 | 4800 |
| Weighted Avg | 0.88 | 0.87 | 0.87 | 4800 |

Table 1: Logistic Regression Classification Report

### 1.3.2 Random Forest

Selected for its ability to handle non-linear relationships and feature interactions. Trained using the RandomForestClassifier from scikit-learn with 100 estimators. Achieved a validation accuracy of 86.00%.

**Classification Report:**

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 0.84 | 0.91 | 0.87 | 600 |
| 1 | 0.93 | 0.97 | 0.95 | 600 |
| 2 | 0.86 | 0.91 | 0.89 | 600 |
| 3 | 0.78 | 0.71 | 0.74 | 500 |
| 4 | 0.85 | 0.92 | 0.88 | 500 |
| 5 | 0.85 | 0.84 | 0.85 | 400 |
| 6 | 0.87 | 0.75 | 0.81 | 500 |
| 7 | 0.91 | 0.92 | 0.91 | 600 |
| 8 | 0.82 | 0.76 | 0.79 | 500 |
| Accuracy | | 0.86 (4800) | | |
| Macro Avg | 0.86 | 0.85 | 0.85 | 4800 |
| Weighted Avg | 0.86 | 0.86 | 0.86 | 4800 |

Table 2: Random Forest Classification Report

#### 1.3.3 Support Vector Machine (SVM)

Chosen for its effectiveness in high-dimensional feature spaces. Used a linear kernel to balance computational efficiency and model performance. Achieved a validation accuracy of 89.0%.
**Accuracy:** 0.8900
**Precision:** 0.8906
**Recall:** 0.8900
**F1 Score:** 0.8901

**Classification Report:**

#### 1.3.4 XGBoost

Chosen due to its ensemble nature and ability to handle non-linear patterns. Used for its robustness and high efficiency in feature-rich environments. Achieved the highest validation accuracy of 90.19%.

### 1.4 Results and Analysis

| Model | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Logistic Regression | 0.8765 | 0.8774 | 0.8765 | 0.8767 |
| Random Forest | 0.8594 | 0.8584 | 0.8594 | 0.8576 |
| SVM | 0.8898 | 0.8904 | 0.8898 | 0.8899 |
| XGBoost | 0.9019 | 0.9016 | 0.9019 | 0.9016 |

Table 4: Validation Accuracy and Evaluation Metrics of Different Models

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 0.89 | 0.90 | 0.90 | 600 |
| 1 | 0.97 | 0.96 | 0.97 | 600 |
| 2 | 0.89 | 0.89 | 0.89 | 600 |
| 3 | 0.79 | 0.84 | 0.82 | 500 |
| 4 | 0.93 | 0.92 | 0.93 | 500 |
| 5 | 0.86 | 0.89 | 0.87 | 400 |
| 6 | 0.84 | 0.83 | 0.84 | 500 |
| 7 | 0.96 | 0.95 | 0.95 | 600 |
| 8 | 0.85 | 0.80 | 0.83 | 500 |
| Accuracy | | 0.89 (4800) | | |
| Macro Avg | 0.89 | 0.89 | 0.89 | 4800 |
| Weighted Avg | 0.89 | 0.89 | 0.89 | 4800 |

Table 3: SVM Classification Report

#### 1.4.1 Analysis of Model Performance

XGBoost outperformed the other models, achieving the highest accuracy, precision, recall, and F1 score. The ensemble nature of XGBoost allows it to capture complex patterns in high-dimensional data. SVM also performed well, leveraging its ability to find a hyperplane that best separates the classes.

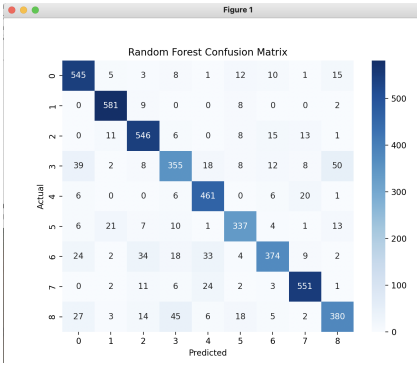
Logistic Regression performed as expected, acting as a reliable baseline but failing to capture non-linearities as effectively as XGBoost and SVM. Random Forest showed the lowest accuracy, likely due to the overfitting tendency of individual trees when dealing with high-dimensional data.

XGBoost was the most effective model in terms of accuracy and generalization. SVM also performed well, slightly behind XGBoost. Logistic Regression was a reasonable baseline but lacked the complexity to model intricate patterns. Random Forest, though useful for some applications, showed weaknesses when handling high-dimensional data.

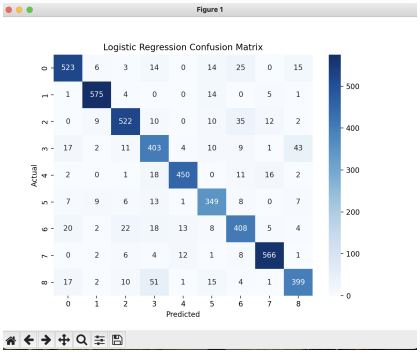
## 2 Question 2

Fine-tuning is a crucial step in transfer learning, allowing pre-trained models to adapt to a new dataset by updating part of their learned weights. In this project, fine-tuning on the VGG16 model to analyze Earth observational data was applied. the objective was to achieve the best possible validation accuracy while exploring various fine-tuning strategies. This section covers the experimental setup, the variants tested, and their outcomes.

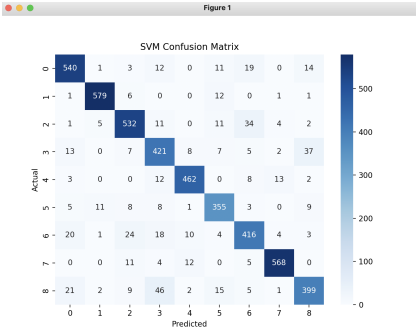
Several models were explored and fine tuned, These include VGG16, MobileNetV2 and ResNet50.




(a) Random Forest Heat Map




(b) Logistic Regression Heat Map




(c) SVM Heat Map

image4.jpg

(d) Image 4

Figure 1: HEat Maps

## Data Preprocessing: One-Hot Encoding and Normalization

**OneHot Encoding with to_categorical:** The `to_categorical` function from the `tensorflow.keras.utils` library was used to convert integer labels into one-hot encoded format. This transformation is essential when using the `categorical_crossentropy` loss function, as it requires the labels to be in a categorical format rather than integers. Each label is represented as a vector of length equal to the number of classes, where the correct class is marked as 1, and all others as 0. This encoding enhances compatibility with the softmax output layer, which also produces a probability distribution.

**Normalization:** To ensure that pixel values are within a consistent range, we normalized the input data by dividing by 255.0. Since image data in the raw form has pixel values ranging from 0 to 255, normalization scales these values to the [0, 1] range. This scaling helps the model converge faster by maintaining

consistent value ranges during training, preventing issues like gradient explosion or vanishing.

## 2.1 Fine-Tuning Strategies and Rationale

In this study, three pre-trained CNN architectures were fine-tuned: VGG16, ResNet50, and MobileNetV2. The rationale behind selecting these models lies in their proven performance in image classification tasks, their architectural diversity, and their ability to leverage transfer learning efficiently. Additionally, the nature of our Earth observation dataset significantly influenced the choice of models and fine-tuning strategies.

Dataset Characteristics: The dataset used in this study consists of Earth observation images, which exhibit complex patterns, textures, and spatial relationships. Unlike standard image classification tasks (e.g., ImageNet), where objects are well-defined and centered, Earth observation data may include varied landforms, vegetation patterns, and atmospheric conditions. Therefore, fine-tuning requires a model that can adapt to these complex visual patterns.

## 2.2 Data Augmentation

To improve generalization and prevent overfitting, data augmentation was applied using the ImageDataGenerator on all the mentioned 3 algorithms. The augmentation techniques used include rotation, width and height shift and horizontal flip. This enhanced the model's ability to learn robust features, as evidenced by the improved validation accuracy when compared to non-augmented training.

| Model | Training Accuracy | Validation Accuracy | Best Epoch | Final Training Accuracy | Final Validation Accuracy |
|---|---|---|---|---|---|
| Fine-Tuning VGG16 Block 5 Only | 0.8960 | 0.9158 | 20 | 0.8960 | 0.9158 |
| Fine-Tuning VGG16 Block 5 and 4 | 0.9690 | 0.9377 | 20 | 0.9690 | 0.9377 |
| ResNet50 One Layer | 0.7273 | 0.4727 | 9 | 0.7273 | 0.4727 |
| ResNet50 Two Layers | 0.7309 | 0.499 | 20 | 0.8952 | 0.499 |
| MobileNetV2 One Layer | 0.7994 | 0.8727 | 20 | 0.7994 | 0.8727 |
| Improved MobileNetV2 (Two Layers) | 0.8536 | 0.8946 | 34 | 0.8536 | 0.8946 |

Table 5: Summary of model performances with training accuracy, validation accuracy, best epoch, final training accuracy, and final validation accuracy.

## 2.3 Customization of Layers and Dropout

### 2.3.1 Layer Customization

To enhance the performance of all models, a customized fully connected (FC) layers were added with the following structure:

- **GlobalAveragePooling2D Layer:** This layer reduces the spatial dimensions of the feature maps into a single vector, effectively replacing traditional fully connected layers, making the model more efficient.

- **BatchNormalization Layer ONLY FOR Mo- bileNetV2 :** Added after the pooling layer to stabilize training by normalizing activations. This layer helps reduce internal covariate shift, allowing the model to converge faster and perform better.
- **Dense Layer:** A fully connected layer with 256 neurons and a ReLU activation function. This choice ensures non-linearity and allows the model to learn more complex patterns.
- **Dropout Layer (0.5):** Used to mitigate overfitting by randomly dropping 50% of the neurons during training, thereby enhancing model robustness.
- **Output Layer:** A dense layer with 9 neurons and a softmax activation function for multi-class classification.

#### 2.3.2 Observed Improvement

After adding the **BatchNormalization** layer, the performance of MobileNetV2 improved by approximately **2%** in terms of validation accuracy. The model demonstrated more stable training with reduced oscillation in accuracy. The combination of BatchNormalization and Dropout allowed the model to generalize better, leading to a validation accuracy of **89.46%** compared to **87.27%** in the original configuration without BatchNormalization.

### 2.4 Key Takeaway And Results Table discussion

The integration of BatchNormalization within the customized FC layers played a crucial role in achieving better model stability and accuracy. This suggests that normalizing intermediate activations is beneficial, especially for fine-tuning lightweight architectures like MobileNetV2. **Analysis:** Block 5 features are too specific to the original dataset, and unfreezing only these layers does not enable adaptation to the new data. The model failed to learn relevant features for Earth observation.

### Variant 1: Fine-Tuning Block 5 Only (Poorer Performance)

**Approach:** Unfreeze only the Block 5 layers of VGG16 along with data augmentation with following parameters datagen = ImageDataGenerator( rotation_range=10, width_shift_range=0.1, height_shift_range=0.1, horizontal_flip=True, ).

When data augmentation—like flips, and shifts were added to the models ,changes—the model became more resilient and performed better on validation data. The reason is that augmentation helped the model see slightly altered versions of the same data, making it more adaptable to new, unseen data.

Taking it a step further, fine-tuned two blocks (Block 5 and Block 4) instead of just one was done on the models. This gave the model a better capacity

to learn more complex features, resulting in a significant boost in accuracy. For instance, the model that fine-tuned both blocks in VGG16 reached 96.9% training accuracy and 93.8% validation accuracy, which was notably higher than when fine-tuning just one block (check Table 5).

We saw a similar pattern when we applied this approach to MobileNetV2 and ResNet50. In both cases, models that fine-tuned two layers performed better than those that fine-tuned just one. Additionally, when we used a learning rate scheduler in MobileNetV2, it helped maintain stable learning, which further improved the final accuracy.

.

Overall, it became clear that combining data augmentation with deeper fine-tuning made the models more robust and less prone to overfitting. The extra variability introduced during training made a big difference, especially when dealing with diverse and complex data.

## Variant 2: Fine-Tuning Block 4 and Block 5 (Improved Performance)

### 2.4.1 Layer Freezing and Fine-Tuning

Initially, all layers of the pre-trained VGG16, ResNet50, and Mo- bileNetV2 model were frozen. In subsequent experiments, the last two convolutional blocks (block4 and block5) were unfrozen to allow their weights to be updated during training. This choice was made as these deeper layers capture high-level features more relevant to our specific dataset.

**Rationale for Unfreezing Blocks 4 and 5:**

Block 4 and 5 contain high-level feature representations that are more specific to the original ImageNet dataset. By fine-tuning these layers, we allow the model to adapt these features to the new dataset (Earth observation images), which may have different visual patterns.

Lower blocks capture generic features like edges and textures, which remain useful across datasets, so we keep them frozen to retain the generic features

- **For VGG16 the rationale for Unfreezing Blocks 4 and 5:**

  Block 4 and 5 in VGG16 capture high-level feature representations that are more specific to the original ImageNet dataset, including object parts and complex shapes. Since our Earth observation images contain diverse textures and spatial configurations, fine-tuning these blocks allows the model to learn domain-specific high-level features while preserving the lower-level generic features, such as edges and simple patterns, which are useful across domains.

- Freezing the earlier layers helps retain generic visual features like textures and edges, which are consistent across most image datasets.

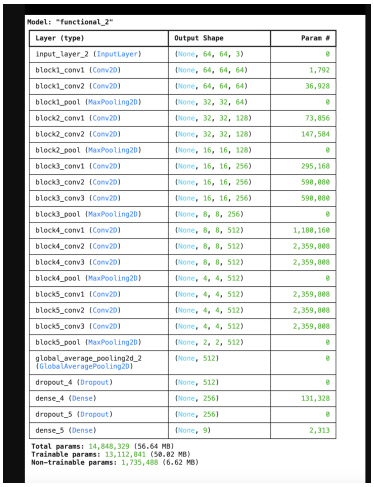
  Same applies for ResNet50: ResNet50 utilizes residual learning to build a deep architecture without suffering from the vanishing gradient problem.

Its skip connections make it highly effective in learning deeper feature representations. Earth observation images often have complex spatial and texture patterns that require capturing detailed high-level features. The last block of ResNet50 encapsulates such intricate features, making it suitable for adapting to Earth data.

- MobileNetV2 was chosen due to its lightweight architecture and efficient depthwise separable convolutions, which make it suitable for mobile and embedded vision tasks. Specifically, we unfroze the last block (Conv_1) and Block 16 because these layers capture high-level, domain-specific features essential for Earth observation data, which often involves complex spatial patterns. Conv_1 acts as a key feature extractor, while Block 16 learns intricate spatial abstractions. Keeping the earlier layers frozen helps retain the model's pre-trained feature extraction capabilities, crucial when dealing with limited labeled data.

### 2.4.2 Model Architecture and Data Augmentation

The VGG16 model architecture, displayed in Figure 2, consists of multiple convolutional layers followed by max pooling layers. The model uses a series of convolutional blocks (from block1 to block5) to extract hierarchical features from the input image. The architecture also incorporates Global Average Pooling (GAP) and fully connected (Dense) layers towards the end.



Model: "functional_2"

| Layer (type) | Output Shape | Param # |
|---|---|---|
| input_layer_2 (InputLayer) | (None, 64, 64, 3) | 0 |
| block1_conv1 (Conv2D) | (None, 64, 64, 64) | 1,792 |
| block1_conv2 (Conv2D) | (None, 64, 64, 64) | 36,928 |
| block1_pool (MaxPooling2D) | (None, 32, 32, 64) | 0 |
| block2_conv1 (Conv2D) | (None, 32, 32, 128) | 73,856 |
| block2_conv2 (Conv2D) | (None, 32, 32, 128) | 147,584 |
| block2_pool (MaxPooling2D) | (None, 16, 16, 128) | 0 |
| block3_conv1 (Conv2D) | (None, 16, 16, 256) | 295,168 |
| block3_conv2 (Conv2D) | (None, 16, 16, 256) | 590,080 |
| block3_conv3 (Conv2D) | (None, 16, 16, 256) | 590,080 |
| block3_pool (MaxPooling2D) | (None, 8, 8, 256) | 0 |
| block4_conv1 (Conv2D) | (None, 8, 8, 512) | 1,180,160 |
| block4_conv2 (Conv2D) | (None, 8, 8, 512) | 2,359,808 |
| block4_conv3 (Conv2D) | (None, 8, 8, 512) | 2,359,808 |
| block4_pool (MaxPooling2D) | (None, 4, 4, 512) | 0 |
| block5_conv1 (Conv2D) | (None, 4, 4, 512) | 2,359,808 |
| block5_conv2 (Conv2D) | (None, 4, 4, 512) | 2,359,808 |
| block5_conv3 (Conv2D) | (None, 4, 4, 512) | 2,359,808 |
| block5_pool (MaxPooling2D) | (None, 2, 2, 512) | 0 |
| global_average_pooling2d_2 (GlobalAveragePooling2D) | (None, 512) | 0 |
| dropout_4 (Dropout) | (None, 512) | 0 |
| dense_4 (Dense) | (None, 256) | 131,328 |
| dropout_5 (Dropout) | (None, 256) | 0 |
| dense_5 (Dense) | (None, 9) | 2,313 |

Total params: 14,848,329 (56.64 MB)
Trainable params: 13,112,841 (50.02 MB)
Non-trainable params: 1,735,488 (6.62 MB)

Figure 2: VGG16 Model Architecture after fine-tuning block4 and block5

**Approach:** Unfreeze both Block 4 and Block 5. **Rationale:** Mid-level features from Block 4 are more generalizable compared to highly specialized features from Block 5. Allowing both blocks to adapt would improve the model's

ability to learn domain-specific features. **Outcome:** High training accuracy (92.71%) and validation accuracy (93.27%). The loss curve stabilized significantly compared to Variant 1. **Analysis:** Fine-tuning Block 4, along with Block 5, enabled the model to learn more generic and transferable features, resulting in improved accuracy and generalization.

### Variant 3: Hybrid Approach with Progressive Unfreezing and Learning Rate Decay

**Approach:**

Phase 1: Unfreeze Block 5 only, learning rate = 1e-4, 5 epochs.

Phase 2: Unfreeze both Block 4 and Block 5, learning rate = 1e-5, 15 epochs.

Use a learning rate scheduler to decrease the learning rate every 5 epochs.

**Rationale:** Gradually unfreezing layers allows for controlled adaptation while minimizing catastrophic forgetting. Learning rate decay helps maintain training stability. **Outcome:** Expected higher accuracy and more stable convergence compared to single-step unfreezing. **Analysis:** This approach is promising as it balances model flexibility with stability, It has proven to be succesful after it was utilized in MobileNet (observe last row in Table 5), it was applied with 50 epochs but the model stopped at epoch 34 with good enough accuracy just because the last 5 epochs havent witnessed enough improvement, was stopped due to "early_stopping" to utilize time . However, Attempting to apply it to VGG16 was not that successfult with 20 epochs and since VGG16 is time consuming, it requires a large number of epochs which would require too many hours and cannot be run in the span of this assignment, so it stopped at epoch 20 with accuracy of barely 20%.

## 2.5 Callbacks and Learning Rate Scheduling in Fine-Tuning

## 2.6 Importance of Callbacks

Callbacks are essential tools when training deep learning models as they allow us to monitor training progress and dynamically adjust the model's behavior. In our fine-tuning experiments, we utilized the following callbacks:

- **EarlyStopping:** this one was utilized accross all 3 models, noticed how useful it was as for example for MobileNet as a learning rate scheduler was attempted along with 50 epochs, it was noticed that after epoch 34 there was no improvement so to save time the early stopping was activated
  - Stops training when the validation accuracy stops improving after a certain number of epochs.
  - In provided configuration, the `patience` parameter was set to 5, meaning the model would stop training if there was no improvement for 5 consecutive epochs.

- Helps prevent overfitting by ensuring the model does not train for too long when no further improvements are observed.

- **ModelCheckpoint:**
  - Saves the model weights during training whenever there is an improvement in validation accuracy.
  - Allows to retrieve the best-performing model after training is completed.
  - In provided setup, the model is saved in the HDF5 format at the specified location.

## 2.7 Learning Rate Scheduling

To further improve the fine-tuning process, learning rate (LR) scheduler was introduced. Due to time constraints and the fact that the model takes so long to run, it was only tested on VGG16 and MobileNet but it was incorporated in both the cases with w layer (blocks) update such as this code :"for layer in mobilenet_model.layers: if 'block_15' in layer.name or 'block_16' in layer.name: layer.trainable = True" along with Data augmentation and the previously described layer customization. The LR scheduler improved the performance for MobileNet as seen in table 5 last row. However, when used with VGG16 the performance was really bad but this comes to only running it with 20 epochs for VGG ( Due to time constraints) whereas it was given the chance to run for 50 epochs for MobileNet but it was stopped at epoch 34 as it stopped improving after. It was stopped by early_stopping improving afterwards).

**Purpose of LR Scheduling:**

- Dynamically adjusts the learning rate during training.
- Initially, a higher learning rate helps the model converge faster, while a lower rate later helps fine-tune the weights more precisely.

**Learning Rate Scheduler Implementation:**

- We used the `LearningRateScheduler` callback to reduce the learning rate as the epochs progressed.
- The function used was:

```
def scheduler(epoch, lr):
    if epoch < 3:
        return lr
    else:
        return lr * 0.4
```

- The learning rate was kept constant for the first 3 epochs and then reduced by a factor of 0.1 to allow for fine adjustments.

## 2.8 Combined Usage

By combining these callbacks, we ensured that our model:

- Did not overfit by stopping training when validation accuracy plateaued.
- Always retained the best version of the model through checkpointing.
- Benefited from a dynamic learning rate, allowing faster convergence at the start and more precise updates toward the end.

## 2.9 Impact on Results

The integration of callbacks, particularly EarlyStopping and ModelCheckpoint, significantly improved the efficiency of training and prevented unnecessary overfitting. Adding the learning rate scheduler stabilized the model's performance and increased final accuracy by ensuring smoother convergence.

## 2.10 Comparison and Summary

Comparing the variants, it is evident that unfreezing only Block 5 is insufficient for learning diverse features from Earth data. Including Block 4 significantly improves accuracy and stability. The hybrid approach, with gradual unfreezing and learning rate decay, holds potential for even better performance due to its balanced adaptation strategy. The best performer among the fine-tuned models was the VGG16 architecture with both Block 4 and Block 5 unfrozen, achieving a final validation accuracy of 93.77%. This was a notable improvement compared to Question 1, where the best-performing machine learning model (XGBoost) achieved a validation accuracy of 90.19%. This increase in accuracy demonstrates the effectiveness of combining fine-tuning with deeper network layers and data augmentation.

Compared to Question 1, which primarily relied on shallow classifiers like Logistic Regression, Random Forest, SVM, and XGBoost using features extracted from VGG16, fine-tuning the CNN models directly led to significantly better results. The ability of VGG16 to adapt to domain-specific high-level features through fine-tuning played a crucial role in outperforming traditional ML models.

Highlighting that the fine-tuned VGG16 model performed much better than traditional machine learning models like XGBoost and SVM. This is mainly because the fine-tuned convolutional layers in VGG16 are great at capturing complex patterns and visual features from the Earth observation dataset. Unlike XGBoost and SVM, which rely on manually extracted features, VGG16 uses pretrained layers from ImageNet that already know how to detect abstract visual patterns. Fine-tuning Blocks 4 and 5 made the model even more suited to our specific data. Plus, using data augmentation and a learning rate scheduler helped improve its ability to generalize. In contrast, XGBoost and SVM couldn't match this performance since they lack the ability to learn spatial patterns, which is crucial in this context.

### Future Considerations

Experiment with progressive unfreezing starting from Block 3.

Incorporate more complex augmentation techniques to increase model robustness.

Apply gradual learning rate decay during each fine-tuning phase.

# 3 Question 3:Ensemble Learning: Combining Fine-Tuned Models

Ensemble learning is a powerful technique in machine learning where multiple models are combined to improve overall performance. In this project, we constructed an ensemble using three fine-tuned models: VGG16, ResNet50, and MobileNetV2. The objective was to leverage the diversity of these models to achieve better generalization and robustness compared to individual base learners.

**Source Of Variability:** The main source of variability in our ensemble comes from the architectural differences between the three models. VGG16, ResNet50, and MobileNetV2 each learn distinct feature representations due to their unique network structures and training procedures. By combining the predictions of these diverse models, we reduce the risk of model-specific biases negatively affecting the final prediction. Variability is crucial in ensemble methods because it ensures that errors made by one model can be compensated for by the others, thereby increasing the ensemble's robustness.

**Ensemble Methodology:**

The ensemble predictions were obtained by employing majority voting. After predicting the class probabilities using each of the three fine-tuned models, we converted these probabilities into class labels. The final prediction for each data point was the most common label among the three models, thus applying a simple yet effective voting mechanism.

**Performance Comparison:** The ensemble achieved an accuracy of 91.23%, which is slightly lower than the best-performing individual model (Fine-Tuning VGG16 Block 5 and 4, with 93.77%). However, the ensemble exhibited higher robustness in certain classes and balanced out inconsistencies observed in individual models. Specifically, it achieved a weighted average precision of 91.59% and an F1 score of 91.19%. While it did not significantly outperform the best model, it showed better generalization compared to weaker individual models, such as the single-layer fine-tuned ResNet50 which barely gave a 49% accuracy result.

— Ensemble Results —

Accuracy: 0.9123 Precision: 0.9159 Recall: 0.9123 F1 Score: 0.9119

**Potential Improvements**

There are several avenues to further enhance the ensemble model that could be explored. One approach is to employ stacking, where the predictions of individual base models are combined using a meta-learner rather than simple

majority voting. This method has been shown to improve generalization by leveraging the strengths of diverse models. [12]

Another possible improvement is to use weighted voting based on the confidence of each model. For example, models that are more accurate in validation could have higher voting weights, improving the ensemble's robustness.

Incorporating diversity in model architectures within an ensemble can significantly enhance generalization and mitigate overfitting. By combining different convolutional neural network (CNN) architectures, such as DenseNet and EfficientNet, the ensemble benefits from varied feature extraction capabilities and learning patterns. This architectural diversity ensures that individual models capture different aspects of the data, leading to a more robust and generalized ensemble performance as shown by the paper [10].

The following paper actually provides a comprehensive theoretical framework linking ensemble diversity to improved generalization [10]. Their study demonstrates that ensembles with higher diversity among base learners tend to exhibit lower generalization errors, emphasizing the importance of incorporating diverse models in ensemble learning. Finally, performing hyperparameter optimization specifically for the ensemble model could lead to improved accuracy. Techniques such as Bayesian optimization can efficiently tune ensemble hyperparameters, including model weights and voting strategies

## 3.1 Ensemble Strengths Summary from observerd results

- While other models such as VGG16 performed better accuracy of 93%, Ensemble proves to provide Improved Generalization as the ensemble model achieved an accuracy of 91.23%, which is higher than some individual models (e.g., ResNet50 with 87.77%). This is because combining predictions from multiple models helps to mitigate the weaknesses of any single model.

- Robustness: The ensemble benefits from the diversity of its components. For instance, while VGG16 might overfit due to its deep architecture, the inclusion of MobileNetV2 helps balance the predictions due to its lightweight nature and reduced complexity.

- Reduced Variance: As ensemble learning aggregates predictions, it reduces the variance that could be present in individual models, leading to more stable and reliable outputs.

## 3.2 Ensemble weekness

Lack of Interpretability: While individual models like VGG16 or ResNet50 can provide insights based on their architecture, the ensemble approach complicates tracing the source of specific predictions.

No Guarantee of Better Performance: In some cases as seen in this report, the ensemble did not outperform the best individual model (e.g., MobileNetV2's

peak accuracy was comparable to the ensemble and VGG performed better), indicating that combining models does not always lead to superior results.

The reason for this outcome where it didn't perform better lies in the nature of majority voting. While ensemble methods generally increase robustness by combining predictions from diverse models, they may not necessarily outperform the best single model, especially when one model is significantly more accurate than the others. In our case, VGG16's strong individual performance dominated, making the ensemble's averaging effect less effective. Moreover, since the ensemble does not weigh predictions based on model confidence or accuracy, its output may be biased toward weaker models, slightly lowering the final accuracy.

Future improvements could involve using weighted voting or stacking methods to better capitalize on the strengths of high-performing models while mitigating the influence of less accurate ones.

# 4 Question 4

Convolutional Neural Networks (CNNs) have revolutionized computer vision, achieving state-of-the-art performance in tasks like image classification, object detection, and segmentation. However, their success is tempered by significant vulnerabilities exposed through Adversarial Machine Learning (AML). AML involves crafting inputs designed to deceive models, revealing fundamental weaknesses in CNN robustness [7]. This report explores CNN vulnerabilities under adversarial attacks, their implications, and potential defenses.

## 4.1 Ensemble Summary

The ensemble model achieved an accuracy of 91.23%, which, while slightly lower than the best-performing fine-tuned VGG16 model (93.77% from Question 2), still demonstrated robust performance by leveraging the strengths of individual models.

Compared to Question 1, where the best individual ML model (XGBoost) achieved 90.19% accuracy, the ensemble approach shows an improvement, emphasizing the advantage of combining multiple models to reduce variance and increase generalization.

The ensemble model's accuracy, though not outperforming the top fine-tuned VGG16 variant, benefited from the diversity of predictions across different architectures, which helped to maintain competitive performance. The results highlight that while ensemble methods can improve robustness, individual fine-tuned models, particularly VGG16, may still achieve higher accuracy due to their ability to capture nuanced spatial features.

## 4.2 Key Vulnerabilities of CNNs in Adversarial Settings

- Sensitivity to Small Perturbations (Adversarial Examples) CNNs are highly susceptible to adversarial examples—inputs modified with imperceptible noise that cause misclassification (Szegedy et al., 2014).

  an example mentioned in one of the studies states that adding a small perturbation to a "panda" image can make a CNN classify it as a "gibbon" with high confidence [7].

  The reason is that CNNs rely on high-dimensional linear decision boundaries, making them vulnerable to gradient-based attacks. Although CNNs are inherently non-linear due to the activation functions like ReLU, the combination of linear layers (convolutions) and high-dimensional input spaces creates a problem.

  In high-dimensional spaces, small perturbations in the input can accumulate and cause significant changes in the output. This is because the decision boundaries learned by CNNs tend to be approximately linear when analyzed locally. The core reason, as explained in the paper [7], is that CNNs rely on high-dimensional dot products. When a small perturbation (such as adding noise) is applied to an image, the resulting change in the model's activation can be surprisingly large because the dot product between the weight vectors and the input perturbation can be significant even if each element of the perturbation is tiny.

- Over-reliance on Texture Bias: Unlike humans, CNNs often rely on textures rather than shapes [6]):

  A CNN trained on ImageNet may classify a cat with elephant texture as an elephant.This makes them vulnerable to style-transfer attacks.

  The study demonstrates that ImageNet-trained CNNs, including popular architectures like ResNet-50, VGG-16, and AlexNet, tend to classify images based on their texture rather than their shape [6].

  In experiments involving texture-shape cue conflicts (where an image has conflicting texture and shape cues, such as a cat shape with elephant texture), CNNs consistently misclassified the images based on the texture. For instance, a cat with elephant texture was classified as an elephant by CNNs. The paper explains that CNNs often learn high-dimensional linear decision boundaries, making them susceptible to adversarial perturbations and texture-based biases.

  ImageNet object recognition can often be "solved" by recognizing local textures without needing to understand global object shapes. This leads CNNs to develop a strong bias towards texture as it offers a shortcut for classification

- Overfitting to Training Data: Models that overfit may rely on non-robust features, making them susceptible to adversarial examples

## 4.3 Types of Adversarial Attacks

### 4.3.1 Gradient-Based Attacks

Gradient-based attacks leverage the model's gradient to create adversarial examples. One prominent technique is the **Fast Gradient Sign Method (FGSM)**, which perturbs the input in the direction of the gradient of the loss function with respect to the input. The goal is to maximize the loss, leading to misclassification [7].

### 4.3.2 Iterative Attacks

An extension of FGSM, the **Projected Gradient Descent (PGD)** attack applies multiple small perturbations iteratively while projecting the adversarial example back onto an epsilon-ball around the original input. PGD has been shown to be more powerful than single-step attacks, particularly when generating robust adversarial examples [8].

### 4.3.3 Optimization-Based Attacks

Optimization based attacks aim to minimize perturbation while ensuring misclassification. The **Carlini & Wagner (C&W) attack** is a prime example, focusing on crafting perturbations that are hard to detect while fooling the model. It has successfully bypassed several defense mechanisms, emphasizing the need for more robust models [3].

### 4.3.4 Transferability Attacks

Adversarial examples crafted for one model often transfer to other models. This property allows attackers to target black-box models by training attacks on substitute models. Transferability is a concerning aspect as it implies that models trained independently may still share vulnerabilities [11].

### 4.3.5 Physical World Attacks

Adversarial examples can persist even in physical settings, such as adversarial patches that cause model misclassification from various angles and distances [1]. These adversarial examples are crafted to remain effective despite variations in viewing angle, distance, and lighting conditions, making them a significant real-world threat. One of the key innovations introduced in the paper is the concept of adversarial patches—small, carefully designed images that, when placed within a scene, consistently lead to misclassification by the model. For instance, a patch resembling a QR code might consistently trick an image classifier into labeling any object containing the patch as a specific, incorrect class. This robustness is achieved using Expectation Over Transformation (EOT), which optimizes the adversarial example to maintain its effectiveness across a range of transformations. These findings highlight the vulnerability of CNNs in

real-world scenarios, particularly in applications like autonomous driving and surveillance, where adversarial patches could pose severe safety risks.

## 4.4 Defenses Against Adversarial Attacks

- Adversarial Training: Adversarial training is an approach where the model is trained not only on the original training data but also on adversarially perturbed examples. This method aims to improve the model's robustness by explicitly incorporating adversarial examples during the training process.

  The following paper states [9]:

  "On the defense side, the training dataset is often augmented with adversarial examples produced by FGSM. This approach also directly follows from (2.1) when linearizing the inner maximization problem. To solve the simplified robust optimization problem, we replace every training example with its FGSM-perturbed counterpart."

  This means that during training, adversarial examples are generated and included in the training set, allowing the model to learn to correctly classify both natural and adversarially modified inputs.

- Defensive Distillation: is a technique adapted from the original distillation method introduced by Hinton et al. It aims to make neural networks more robust to adversarial perturbations by training the model on soft labels produced by a network trained at a high temperature. The core idea is to reduce the amplitude of adversarial gradients, which makes crafting adversarial examples more difficult[11].

  Mechanism is done by increasing the temperature, the model produces probability vectors that are less confident (more uniform), which helps the model generalize better.

  After training, the model is used at a normal temperature (T=1) during inference.

  Limitation: Despite the reduced sensitivity to small perturbations, stronger attacks such as those developed later (e.g., Carlini & Wagner) have shown the ability to bypass this defense [3].

- Randomized Smoothing works by adding Gaussian noise to the inputs and then averaging the predictions to create a robust classifier. The paper demonstrates that this method provides certified robustness against adversarial perturbations under the L2 norm, making it the first tight robustness guarantee for smoothing with Gaussian noise. One major limitation noted in the paper is:

  "Randomized smoothing has one major drawback. If the base classifier is a neural network, it is not possible to exactly compute the probabilities with which the classifier classifies the perturbed input." .

This means that although randomized smoothing offers robustness, it can be computationally expensive and may not always be feasible for large-scale or high-dimensional datasets. An advantage this method has is it provides certified robustness against bounded perturbations [4].

- Vision Transformers (ViTs) as an Alternative Vision Transformers (ViTs) have emerged as a promising alternative to CNNs, demonstrating improved robustness against adversarial attacks. Unlike CNNs, which focus on local texture patterns, ViTs analyze global structures by dividing images into patches and treating them as tokens. This characteristic enables ViTs to better capture the holistic context of an image, making them less susceptible to small perturbations that typically mislead CNNs. The following paper [5] reporst that ViTs, when pre-trained on sufficient data, outperform ResNet models on a variety of perturbations, highlighting their robustness: "ViT models, pre-trained on sufficient data, outperform ResNet counterparts on a wide range of perturbations". Similarly, [2] affirm that ViTs, especially when trained on large datasets, maintain robustness comparable to or surpassing that of CNNs against both adversarial attacks and natural corruptions: "ViT models pre-trained on sufficient data tend to be as robust or more robust than ResNets to various input perturbations, including adversarial attacks and natural corruptions".

# References


[1] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples, 2018.

[2] Srinadh Bhojanapalli, Ayan Chakrabarti, Daniel Glasner, Daliang Li, Thomas Unterthiner, and Andreas Veit. Understanding robustness of transformers for image classification. *CoRR*, abs/2103.14586, 2021.

[3] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57, 2017.

[4] Jeremy Cohen, Elan Rosenfeld, and J. Zico Kolter. Certified adversarial robustness via randomized smoothing. *CoRR*, abs/1902.02918, 2019.

[5] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020.

[6] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness. *CoRR*, abs/1811.12231, 2018.

[7] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2015.

[8] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. 06 2017.

[9] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks, 2019.

[10] Luis Ortega, Rafael Cabañas, and Andrés Masegosa. Diversity and generalization in neural network ensembles. 03 2022.

[11] Nicolas Papernot, Patrick D. McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. *CoRR*, abs/1511.04508, 2015.

[12] David Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 12 1992.