# Environmental data science - Project Report

CAYETANOT Pierre-Olivier 皮耶爾, TRAN Huynh-Duy, PERAUD Arthur
Instructor: Prof. Whyjay Zheng

***Abstract*—This study addresses the GeoLifeCLEF 2025 challenge of predicting plant species presence using Sentinel-2 and LandSat satellite imagery. We developed multi-label classification models, including CNN, ResNet, LSTM, LandSatCNN, and LandSatMLP, to leverage spatial and temporal patterns. Evaluated on 5 million Presence-Only and 90,000 Presence-Absence records, LSTM achieved the highest F1-score (0.33), while MLP and LandSatCNN outperformed ResNet on Kaggle scores (0.17 vs. 0.127). Temporal models excelled, but further work on merging temporal environmental data and satellite images is needed.**

## I. Introduction

Understanding the distribution of plant species across diverse geographic and environmental conditions is a critical challenge in environmental data science, with applications in biodiversity conservation, ecosystem management, and citizen science. The GeoLifeCLEF 2025 competition, hosted on Kaggle, addresses this challenge by tasking participants with predicting the presence of plant species at specific GPS coordinates and times using multi-modal environmental data. These data include high-resolution satellite imagery, climatic time series, land cover, soil properties, elevation, and human footprint metrics, providing a rich and complex set of predictors for species distribution modeling (SDM). The dataset comprises approximately 5 million Presence-Only (PO) occurrences and 90,000 Presence-Absence (PA) survey records, covering around 10,000 species of European flora. Figure 1) shows the overall goal of the competition.

This project aims to develop robust multi-label classification models to predict plant species presence based on the provided multi-modal data. Given the dataset's diversity—encompassing spatial satellite image patches, temporal climatic and satellite time series, and environmental rasters—we selected a combination of machine learning and deep learning approaches tailored to the data's characteristics. Specifically, we employ convolutional neural networks (CNNs) to extract spatial features from satellite imagery as well as recurrent neural networks (RNNs) to model temporal dynamics. These model choices leverage the dataset's high-dimensional spatial and temporal features to improve prediction accuracy, as measured by the competition's micro-averaged F1-score. By addressing this challenge, we aim to contribute to high-resolution biodiversity mapping and enhance tools like Pl@ntNet for species identification, aligning with course objectives.

*a) Motivation for Model Choices::*

- **Convolutional Neural Network (CNN)**: CNNs are well-suited for processing LandSat images due to their ability to extract hierarchical spatial features, such as vegetation patterns and land cover variations, from multi-band imagery. Their convolutional layers effectively capture local correlations in pixel values, which are critical for identifying environmental conditions influencing species presence.
- **Residual Network (ResNet)**: ResNet, specifically an 18-layer architecture, is chosen for its ability to handle the high-resolution Sentinel-2 and LandSat imagery while mitigating training challenges.
  - **Feature Preservation**: Skip connections maintain low-level spectral signatures critical for land cover classification, ensuring that subtle vegetation signals in the four-band Sentinel-2 data are not lost during deep feature extraction.
  - **Gradient Flow**: Skip connections enable training of deeper networks on limited satellite data (typically $< 10^4$ samples after preprocessing), addressing vanishing gradient issues and improving convergence.
  - **Multi-Scale Processing**: The 18-layer depth provides a sufficient receptive field for 30 m-resolution pixels (e.g., LandSat), capturing broader spatial contexts without overfitting, which is essential for modeling diverse European landscapes.
- **Long Short-Term Memory (LSTM)**: LSTMs are selected to model the LandSat time series, which span 86 seasons (winter 1999 to autumn 2020) across six bands. LSTMs excel at capturing long-term temporal dependencies, such as seasonal vegetation cycles and land use changes, which are critical for predicting species distributions influenced by historical environmental dynamics.
- **Temporal CNN (`LandsatCNN`)**: The `LandsatCNN` model, a custom convolutional neural network designed for LandSat time series, is chosen for its ability to process the six-band (R, G, B, NIR, SWIR1, SWIR2) time series data reshaped as 2D inputs (6 channels, 86 seasons). Its architecture, consisting of two convolutional layers with 3×3 kernels followed by max-pooling and fully connected layers, is motivated by the following:
  - **Temporal Feature Extraction**: The convolutional layers capture local temporal patterns (e.g., seasonal variations within a few time steps) across the six spectral bands, enabling the model to identify short-term vegetation dynamics relevant to species presence.
  - **Compact Architecture**: With only two convolutional layers and a reduced feature map size (via max-pooling), `LandsatCNN` is computationally efficient, suitable for the high-dimensional time series data (6×86 per observation) and the limited sample size after preprocessing ($< 10^4$ samples).

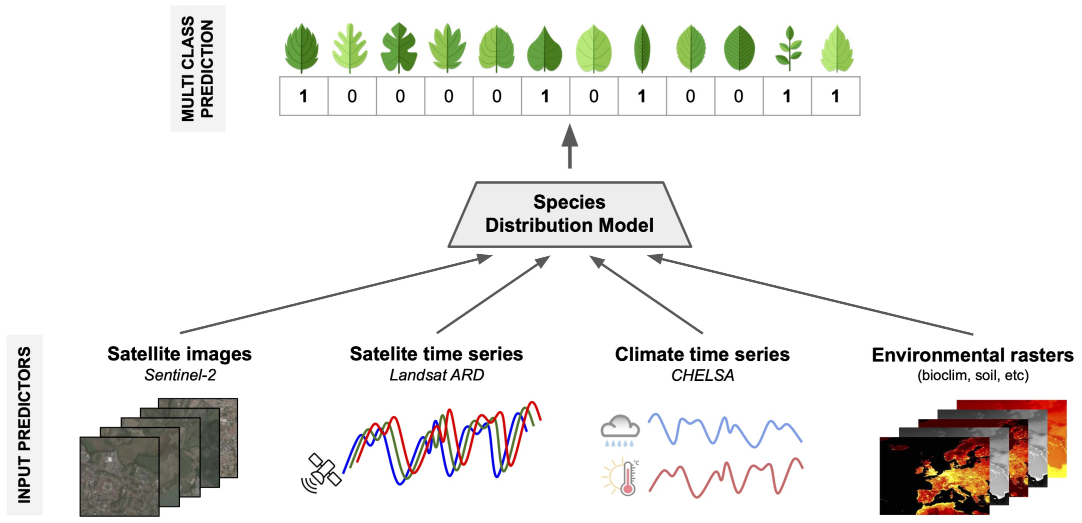

Fig. 1. GeoLifeCLEF 2025 Goal from the Kaggle competition. Available at https://www.kaggle.com/competitions/geolifeclef-2025

- **Multi-Band Integration**: The model processes all six bands simultaneously as input channels, preserving inter-band correlations (e.g., between NIR and SWIR for vegetation health) that are critical for ecological modeling.

- **Multi-Layer Perceptron (`MLP`)**: The `MLP` model, a multi-layer perceptron originally inspired by kernel-based methods, is designed to process the LandSat time series reshaped as a high-dimensional input ($6 \times 4 \times 21$, corresponding to 6 bands, 4 quarters, and 21 years). Its architecture, featuring layer normalization followed by three fully connected layers, is motivated by the following:
  - **High-Dimensional Processing**: The MLP's ability to handle flattened high-dimensional inputs (504 features after reshaping) allows it to capture complex relationships across all bands and time steps, akin to kernel-based feature transformations, without requiring explicit kernel computations.
  - **Stabilizing Training**: Layer normalization before the fully connected layers mitigates issues with varying scales in the LandSat time series (e.g., differing ranges across R, NIR, and SWIR bands), improving training stability on the limited dataset ($< 10^4$ samples).
  - **Simplicity and Flexibility**: The sequential architecture with ReLU activations and progressively reduced layer sizes (1024 to 512 to 10,000 classes) provides a computationally efficient alternative to convolutional models, suitable for rapid prototyping and ensemble integration with other models.

## II. Methods

### A. Data Description

The GeoLifeCLEF 2025 competition, hosted on Kaggle, provides a comprehensive dataset for predicting plant species presence at specific GPS coordinates across Europe. The dataset comprises approximately 5 million Presence-Only (PO) occurrences and around 90,000 Presence-Absence (PA) survey records, covering roughly 10,000 species of European flora. For this project, we focus exclusively on the satellite imagery components, namely Sentinel-2 image patches and LandSat time series, to learn how to handle satellite data. These data, sourced from the Ecodatacube platform, offer high-resolution insights into vegetation dynamics and environmental conditions, which are critical for our convolutional neural network (CNN) and recurrent neural network (RNN) approaches. Below, we describe the species observation data and satellite imagery components in detail.

*Species Observation Data:* The dataset includes two types of species observation records, which provide the ground-truth labels for training our multi-label classification models:

- **Presence-Absence (PA) Surveys**: Approximately 90,000 survey records document the presence or absence of around 10,000 plant species. These data are used to calibrate models and mitigate biases in PO data, particularly false absences due to incomplete sampling.
- **Presence-Only (PO) Occurrences**: Approximately 5 million opportunistically sampled observations from the Global Biodiversity Information Facility (GBIF). These records cover a wide geographic range but are subject to sampling biases, as the absence of a species does not confirm its true absence due to non-standardized sampling protocols.

Each observation is associated with a unique `surveyId`, linking it to corresponding satellite imagery data for precise geolocation-based analysis. The PA data are critical for validating model predictions, while the PO data provide a larger, albeit noisier, training set.

*Satellite Imagery Data:* We utilize two types of satellite imagery data: Sentinel-2 image patches for spatial analysis and LandSat time series for temporal analysis, described below.

The Sentinel-2 data consist of high-resolution image patches centered at each observation's geolocation, capturing spatial environmental features relevant to plant species distributions.

- **Format**: 64×64 pixel TIFF files, each containing four bands: Red (R), Green (G), Blue (B), and Near-Infrared (NIR). Tensors are also made available.
- **Resolution**: 10 meters per pixel, covering a 640m×640m area per patch.
- **Temporal Scope**: Each patch is taken from the same year as its corresponding observation, ensuring temporal alignment with species records.
- **Source**: Sentinel-2 remote sensing data

These image patches are well-suited for CNNs, as their high spatial resolution and multi-band structure capture local vegetation patterns, phenological characteristics, and land cover variations that influence species presence.

The LandSat data provide temporal dynamics of environmental conditions at each observation location, capturing seasonal vegetation changes over two decades.

- **Format**: Tensors with three axes: bands (Red, Green, Blue, NIR, SWIR1, SWIR2), season (median over 3 months) and year.
- **Resolution**: 30 meters per pixel, derived from the original LandSat satellite data.
- **Temporal Scope**: Seasonal medians from winter 1999 to autumn 2020.

These time series are ideal for modeling temporal patterns, such as seasonal vegetation cycles or land use changes, which are critical for predicting species distributions using RNNs or temporal CNNs.

## B. Dataset and preprocessing

To download the data, we need to install the kaggle Python package. After that, executing the following script will execute the command and unzip the data.

```python
!kaggle competitions download -c geolifeclef-2025 -p ../data

import zipfile
with zipfile.ZipFile("../data/geolifeclef-2025.zip", 'r') as zip_ref:
    zip_ref.extractall("../data")
```

Listing 1. Python code for downloading the competition data

For Sentinel-2 images, quantile normalization is used to preprocess our data.

```python
def quantile_normalize(band, low=2, high=98):
    sorted_band = np.sort(band.flatten())
    quantiles = np.percentile(sorted_band, np.linspace(low, high, len(sorted_band)))
    normalized_band = np.interp(band.flatten(), sorted_band, quantiles).reshape(band.shape)

    min_val, max_val = np.min(normalized_band), np.max(normalized_band)

    # Prevent division by zero if min_val == max_val
    if max_val == min_val:
        return np.zeros_like(normalized_band, dtype=np.float32)  # Return an array of zeros

    # Perform normalization (min-max scaling)
    return ((normalized_band - min_val) / (max_val - min_val)).astype(np.float32)
```

Listing 2. Quantile normalization by Lukáš Picek

## C. Residual Networks (ResNet)

*1) Introduction to ResNets:* Residual Networks (ResNet) [1] revolutionized deep learning by introducing *skip connections* that solve the vanishing gradient problem in very deep networks. The core innovation is the residual block:

$$\mathbf{y} = \mathscr{F}(\mathbf{x}, \{W_i\}) + \mathbf{x} \tag{1}$$

where $\mathbf{x}$ is the input feature map, $\mathscr{F}$ represents the residual mapping to be learned, and $\{W_i\}$ are the weights of convolutional layers. When the input and output dimensions differ, a linear projection $W_s$ is applied:

$$\mathbf{y} = \mathscr{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x} \tag{2}$$

Each residual block typically contains two $3 \times 3$ convolutional layers with batch normalization and ReLU activation:

$$\mathscr{F}(\mathbf{x}) = W_2\sigma(W_1\mathbf{x}) \tag{3}$$

where $\sigma$ denotes the ReLU function $\max(0, x)$. The complete ResNet-18 architecture (shown in Figure 2) stacks four such blocks with feature map downsampling.

*a) Motivation for Satellite Imagery::*

- **Feature Preservation**: Skip connections maintain low-level spectral signatures critical for land cover classification
- **Gradient Flow**: Enables training of deeper networks on limited satellite data (typically $<10^4$ samples)
- **Multi-Scale Processing**: The 18-layer depth provides sufficient receptive field for 30m-resolution pixels (e.g., Landsat) without overfitting

*2) Modified ResNet-18 Architecture:* The implementation builds upon the ResNet-18 architecture pretrained on ImageNet, with three critical modifications for multispectral satellite imagery classification:

1) **Input Layer Adaptation**: The first convolutional layer is modified to accept 4-channel input (typical for multispectral data):

$$\text{conv1} = \text{Conv2d}(4, 64, \text{kernel\_size} = (7,7), \text{stride} = (2,2)) \tag{4}$$

replacing the original 3-channel RGB input layer while preserving the spatial processing characteristics.

2) **Feature Extractor**: The pretrained residual blocks remain unchanged, maintaining the original skip connections:

$$\mathbf{y}_l = \mathscr{F}_l(\mathbf{x}_l, \{W_i^l\}) + \mathbf{x}_l \quad \text{for } l \in \{1, \dots, 4\} \tag{5}$$

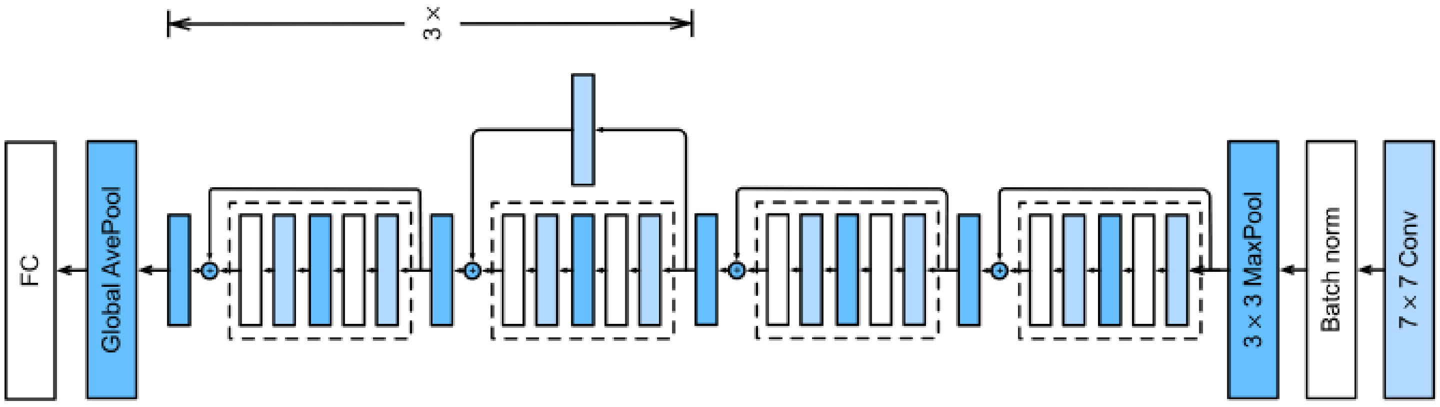

Fig. 2. Resnet-18 architecture [2]

where $l$ indexes the four residual stages with increasing channel depths $\{64,128,256,512\}$.

3) **Classification Head**: The final fully-connected layer is replaced for multi-label prediction:

$$\text{fc} = \text{Linear}(512, N_{\text{classes}}) \tag{6}$$

This implementation is introduced by Lukáš Picek as a baseline for the competition. This approach as well as the preprocessing over the Sentinel-2 data is used as a baseline to compare our models.

*3) LandSat CNN:* To process the LandSat time series, we implemented a custom temporal convolutional neural network (`LandsatCNN`) using PyTorch. The model takes the six-band time series (R, G, B, NIR, SWIR1, SWIR2) over 86 seasons, reshaped as a 2D input (6 channels, 86 time steps). The code below defines the model architecture, featuring two convolutional layers with max-pooling and fully connected layers for multi-label classification.

```python
import torch
import torch.nn as nn
import torch.nn.functional as F

class LandsatCNN(nn.Module):
    def __init__(self, num_classes):
        super(LandsatCNN, self).__init__()
        # Define the layers
        self.conv1 = nn.Conv2d(6, 32,
            kernel_size=3, stride=1, padding
            =1)
        self.conv2 = nn.Conv2d(32, 64,
            kernel_size=3, stride=1, padding
            =1)
        self.pool = nn.MaxPool2d(kernel_size
            =2, stride=2)
        self.fc1 = nn.Linear(64 * 1 * 5,
            128)
        self.fc2 = nn.Linear(128,
            num_classes)

    def forward(self, x):
        # Reshape the input to [batch_size,
            channels, height, width]
        x = x.permute(0, 1, 3, 2)
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 64 * 1 * 5)   #
            Flatten the tensor
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

Listing 3. Python code for LandsatCNN model

The `LandsatCNN` model in Listing 3 processes the pre-processed LandSat time series by applying two convolutional layers with 3×3 kernels to extract temporal features, followed by max-pooling to reduce dimensionality and fully connected layers to produce species presence probabilities. The model's design balances computational efficiency with the need to capture temporal patterns across the six spectral bands.

*4) LandSat MLP:* To process the LandSat time series, we implemented a Multi-Layer Perceptron (`LandSatMLP`) using PyTorch, originally inspired by kernel-based methods but adapted for simplicity and efficiency. The model takes the six-band (R, G, B, NIR, SWIR1, SWIR2) time series over 84 seasons (reshaped as 6×4×21 for 6 bands, 4 quarters, and 21 years) and outputs predictions for 10,000 plant species. The code below defines the model architecture, featuring layer normalization followed by three fully connected layers with ReLU activations.

```python
import torch
import torch.nn as nn
import numpy as np

class LandSatMLP(nn.Module):
    def __init__(self, num_classes,
        input_shape=(6, 4, 21)):
        super(LandSatMLP, self).__init__()
        flat_size = np.prod(input_shape)
        self.model = nn.Sequential(
            nn.Flatten(),
            nn.LayerNorm(flat_size),
            nn.Linear(flat_size, 1024),
            nn.ReLU(),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Linear(512, num_classes)
        )

    def forward(self, x):
        return self.model(x)
```

Listing 4. Python code for LandSatMLP model

The `LandSatMLP` model in Listing 4 processes the preprocessed LandSat time series by flattening the high-dimensional input (6×4×21) and applying layer normalization to stabilize training. The three fully connected layers with ReLU activations reduce the dimensionality from 504 features to 1024, 512, and finally 10,000 classes, producing species presence probabilities. The model's simplicity and layer normalization make it suitable for rapid prototyping and ensemble integration with other models like `LandsatCNN`.

## *D. LSTM model*

*1) Model explanations:* The LSTM algorithm is an advanced method developed from the neural network. The neural network (NN) is also known as an Artificial Neural Network (ANN), and it is inspired by how biological neural networks in the human brain process information. Each link can send a signal from one artificial neuron to another.

A typical ANN structure will consist of three primary layers. The input layer will assign a weight value to each input feature. The features and their weight will then be sent to the hidden layer. This layer includes a neuron that computes the total of all input features ($x_i$) by multiplying their weight value ($W_i$) by the function below:

$$\sum_{i=1}^{n} x_i W_i \tag{7}$$

where $n$ is the total number of input features, and $x$ and $W$ denote the feature and its weight. The outcome will be passed to the activation function. The sigmoid function, also known as the tanh function, is commonly employed. They will then proceed to the output layer to generate the prediction results.

The Long Short-term Memory (LSTM) network was proposed by Hochreiter & Schmidhuber (1997). This type of network is typically utilized for data that has a sequential significance. Because it is an advanced network from the ANN, the LSTM also contains three essential layers: the input layer, the hidden layer, and the output layer. Unlike the ANN, LSTM employs new features and previous data to anticipate the result. This is the reason for the improvement in prediction. Moreover, the input features need to have a sequential meaning.

Figure 3 illustrates the general LSTM algorithm's structure. $C_{t-1}$ and $h_{t-1}$ are the historical results obtained from the initial state; $x_t$ is the new feature input. Furthermore, the LSTM will create three values: $h_t$ and $C_t$, which are used to forecast the next state, and $y_t$, which is the prediction result obtained from the present state. As shown in the picture, one neuron includes $f_t$, $i_t$, $\tilde{C}_t$, and $o_t$ data. The $f_t$ value is the value from the forget gate, which is responsible for determining which features from the previous state are significant and utilized to anticipate the output. $i_t$ denotes the value from the input gate that contributes to identifying which features from the fresh input set are required for predicting the result. $\tilde{C}_t$ value will respond by adding the filtered features from the new input and initial state to the current cell state for creating the output for the cell state. In this gate, the LSTM will generate the $C_t$ value for the next state, which will be sent into the output gate and combined with the output gate value $o_t$ to produce $h_t$. This output gate will activate the weight and determine which feature is reasonable and necessary for the following state to utilize prediction. Furthermore, the $h_t$ is used to calculate the forecast outcome $y_t$ of this condition.

Inside each gate, we will notice that the $f_t$, $i_t$, $\tilde{C}_t$, and $o_t$ are computed using the following functions:

$$f_t = a(W_{xf} \times x_t + W_{hf} \times h_{t-1} + b_f) \tag{8}$$

$$i_t = a(W_{xi} \times x_t + W_{hi} \times h_{t-1} + b_i) \tag{9}$$

$$\tilde{C}_t = a(W_{xc} \times x_t + W_{hc} \times h_{t-1} + b_c) \tag{10}$$

$$O_t = a(W_{xo} \times x_t + W_{ho} \times h_{t-1} + b_o) \tag{11}$$

where $W_{xf}$ and $W_{hf}$ are the weights from input-to-forget and state-to-forget, $W_{xi}$ and $W_{hi}$ are the weights from input-to-input and state-to-input, $W_{xo}$ and $W_{ho}$ is the weights from input-to-output and state-to-output, $W_{xc}$ and $W_{hc}$ are the weights and bias used for cell state; $b_f$, $b_i$, $b_c$, and $b_o$ are the biases for forgetting gate, input gate, cell state, and output gate, respectively; $x_t$ and $h_{t-1}$ is the new features and features from the previous state; $a$ is the activation function. This activation function is used to activate the weight of each input feature, supporting the algorithm to predict the output. Moreover, the activation function also helps the algorithm limit neuron output. This will prevent the computational issue when the magnitude of the output becomes too large.

In general, the previous information passes through the forgetting gate, input gate, cell state, and output gate to generate $f_t$, $i_t$, $\tilde{C}_t$, and $o_t$ values. After that, the model will use these values to create the output information (which is $C_t$) by the following function.

$$C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t \tag{12}$$

where $C_{t-1}$ is also the information from the previous state. Then the $C_t$ will be the result for this state. Another piece of information that will be used for the next state is $h_t$, and this value is generated by the function below:

$$h_t = o_t \times a(C_t) \tag{13}$$

After the model generates $h_t$, this value is also used to get the predicted result, which is $y_t$

$$y_t = W_y \times h_t + b_y \tag{14}$$

where $W_y$ is the weight for output, and $b_y$ is the bias for the output.

The same as any other network, the primary purpose of LSTM is to optimize the difference between the prediction and observation values by assessing the loss function, mean-squared error (MSE):

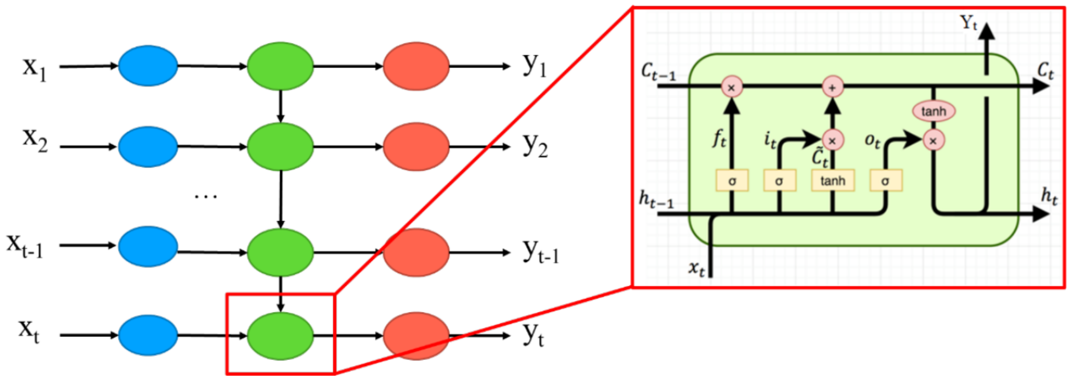

Fig. 3. The basic structure of a Long short-term memory network.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{15}$$

Where $n$ is the total number, $y_i$ is the prediction value from the model, and $\hat{y}_i$ is the observation value.

## III. Results

### A. LSTM Fine Tuning and Results

*1) Number of Epochs:* An epoch finishes when an entire dataset is passed through the neural network. In the neural network, the weights will be updated several times depending on the number of epochs. Training the model on an optimal number of epochs is essential to increase its generalization ability. A fixed number of epochs can't improve model performance. A neural network model trained over many epochs learns patterns specific to the sample data to a greater degree when the number of epochs is greater than necessary. As a result, the model cannot perform well with a new dataset (the overfitting result – Figure 4). While this model performs well on the training set (sample data), it is not accurate on the test set. In other words, the model loses its ability to generalize when overfitting. On the other hand, if the number of epochs is too small, the model will not have enough time to capture the pattern of the dataset. This will lead to the underfitting result, as shown in Figure 4. Training the neural network for the optimal number of epochs is recommended to minimize overfitting and increase generalization capabilities.

Figure 5 presents the evolution of the model's F1-score and accuracy over training epochs when using a learning rate of 0.0001. The evaluation was performed at intervals of 10 epochs, ranging from epoch 10 to epoch 100. The F1-score, which reflects the harmonic mean of precision and recall, is plotted as a continuous line with circular markers, while accuracy, representing the overall proportion of correctly predicted instances, is marked with diamonds. These two metrics provide complementary perspectives on model performance, particularly important when dealing with imbalanced classification problems. Overall, both performance indicators demonstrate a consistent upward trend throughout the training period, suggesting that the model benefits from extended training under this learning rate. The F1-score increases gradually from approximately 0.07 at epoch 10 to a peak of around 0.29 at epoch 90. Accuracy follows a similar trajectory, beginning near 0.01 and reaching its highest point of about 0.28 also at epoch 90. These improvements indicate that the model continues to learn meaningful features with additional epochs, enhancing its capacity to both correctly classify instances and maintain a balance between precision and recall.

Notably, the F1-score curve exhibits some fluctuations between epochs 30 and 60, where minor plateaus or dips are observed. These could reflect temporary stagnation in the learning process or sensitivity to local minima during optimization. However, after epoch 60, both metrics resume their ascent, culminating in the best observed performance at epoch 90. A slight decline at epoch 100 in both F1-score and accuracy suggests the early signs of overfitting, where continued training may begin to degrade generalization due to excessive model tuning on the training data. The similarity in trends between F1-score and accuracy implies that improvements in classification are not limited to one class or driven solely by dominant classes in the dataset, but rather reflect more balanced learning. This is particularly valuable in applications where misclassification costs are unequal, and ensuring a high F1-score is as critical as maximizing accuracy.

In conclusion, the model trained with a learning rate of 0.0001 achieves its optimal performance at **epoch 90**, with both F1-score and accuracy reaching their respective peaks. Beyond this point, performance gains diminish or slightly regress, highlighting epoch 90 as a suitable stopping point to balance model effectiveness and training efficiency. These findings emphasize the importance of monitoring multiple performance metrics over time and adjusting training duration accordingly to avoid underfitting or overfitting.

*2) Learning Rate ($\alpha$):* Learning Rate ($\alpha$) determines the step size at each iteration while moving toward a minimum loss function. So this parameter also affects the generation of the output prediction. Figure 6 illustrates the impact of the learning rate on the training process. Training will progress very slowly if the learning rate value is too small. This can lead to a lengthy training process, and sometimes this long process can lead to a break during the training. On the other hand, if the learning rate is significant, the model can cause undesirable divergent behavior in the loss function. This means the model

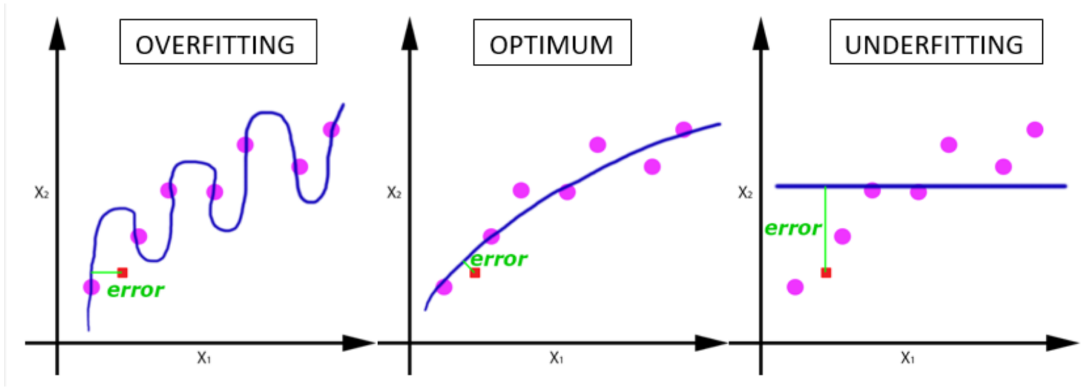

Fig. 4. The underfitting and overfitting issues in the training model

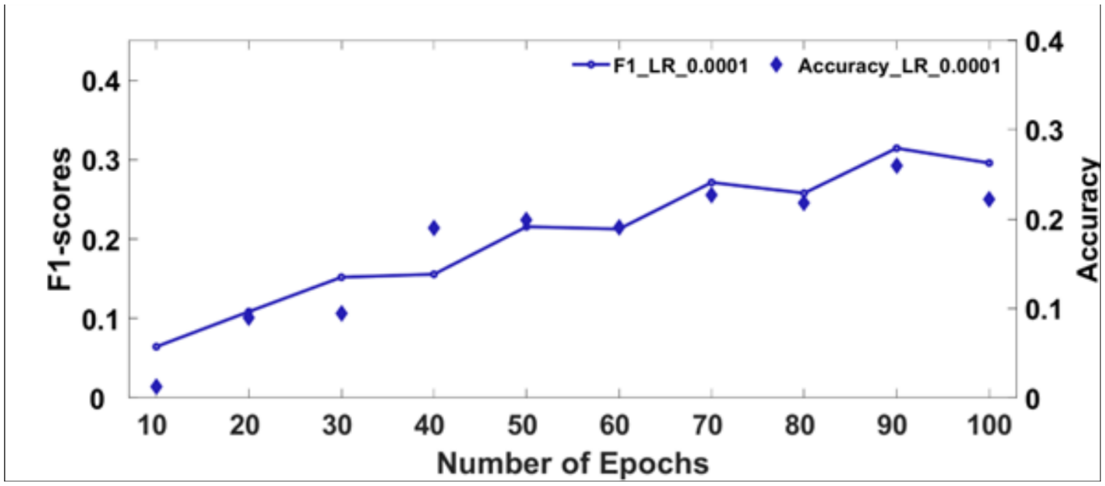

Fig. 5. The statistical result from different numbers of Epochs

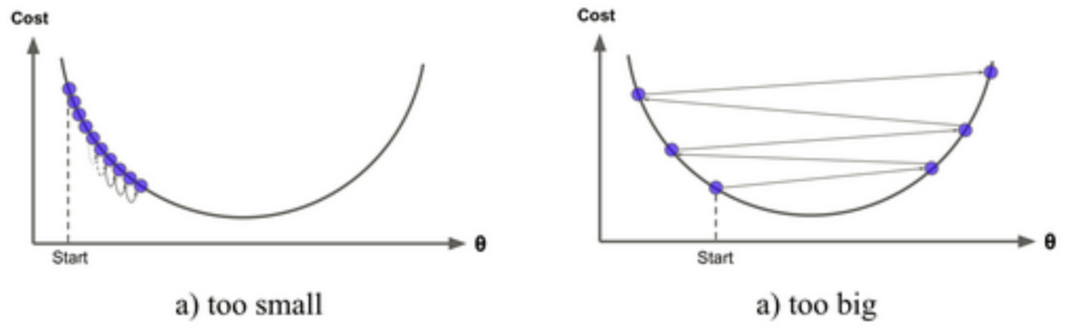

Fig. 6. The effect of the learning rate on the loss function

can "jump over" the optimal value in the loss function. Both parameters cause a significant error and a lousy correlation between the observation and prediction values.

Figure 7 compares the evolution of F1-score and accuracy across different training epochs for two learning rates: 0.0001 and 0.0005. The performance metrics are evaluated every 10 epochs from epoch 10 to epoch 70. The F1-scores are shown as connected line plots with circular markers, while accuracy values are presented using diamond markers. The blue series corresponds to the learning rate of 0.0001, and the orange series represents the results for the higher learning rate of 0.0005. This visualization provides a direct comparison of how learning rate affects convergence speed and model effectiveness over time. For both learning rates, the overall trend shows increasing F1-scores and accuracy with more training epochs, indicating effective learning progression. However, the model trained with the higher learning rate of 0.0005 consistently outperforms the one with 0.0001 in both F1-score and accuracy throughout the training period. Specifically, at epoch 70, the F1-score reaches approximately **0.33** for the 0.0005 setting,

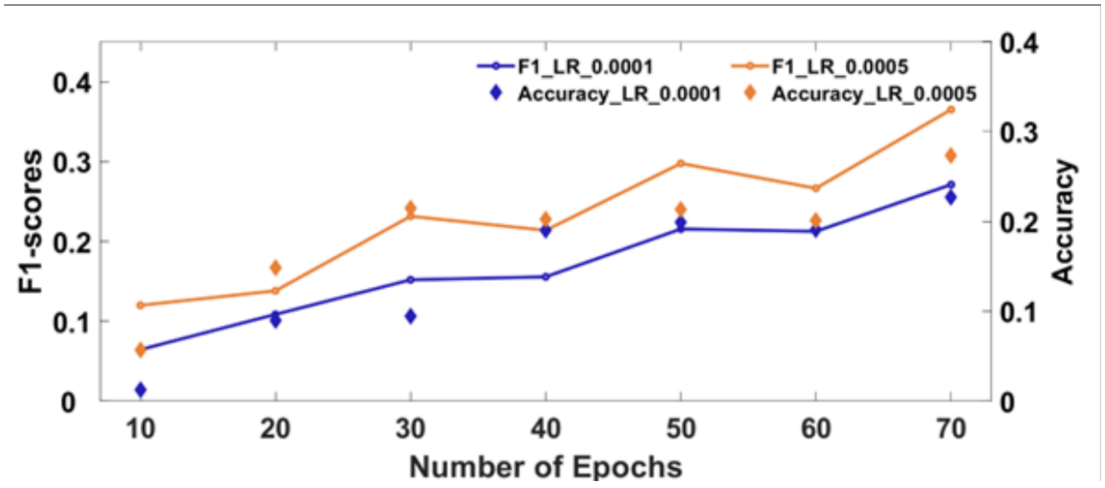

Fig. 7. The statistical result from different numbers of Epochs at 2 different Learning rates.

compared to **0.25** for 0.0001. Similarly, accuracy peaks at about **0.30** for 0.0005, while the lower learning rate achieves around **0.24**.

The faster improvement observed with the 0.0005 learning rate suggests that the model benefits from larger gradient updates, enabling it to escape local minima and converge more quickly within the tested epoch range. Notably, the F1-score for the 0.0005 model shows a steeper slope between epochs 20 and 50, reflecting a more efficient learning phase. This accelerated convergence may be particularly advantageous in scenarios where training time or computational resources are constrained. Despite the overall superior performance of the 0.0005 model, minor fluctuations - especially at epoch 60 - hint at potential instability or over-adjustment in learning. In contrast, the 0.0001 model exhibits a smoother and more gradual improvement, indicating a more stable learning process, albeit slower. This behavior is typical of smaller learning rates, which provide more cautious updates to model weights and reduce the risk of overshooting optimal values but may require longer training to reach comparable performance.

Based on both F1-score and accuracy, the model trained with a learning rate of **0.0005** yields the best overall performance among the configurations tested. While the lower learning rate of 0.0001 produces more stable but slower learning, the 0.0005 setting achieves better convergence within fewer epochs without clear signs of overfitting. Therefore, for the task and dataset considered, a learning rate of 0.0005 with approximately 70 epochs is recommended as the optimal setting for balancing efficiency and classification effectiveness.

*3) Final Result:* Due to the large classification label (SpeciesID), I treated it as the regression task and plot the performance as the linear plot.

Figure 8 displays the classification output of an LSTM-based model trained with 100 hidden units, a learning rate of 0.0005, and 90 training epochs; the y-axis represents the label classification, and the x-axis represents the sample ID. The plot compares the true class labels (blue dashed line) against the predicted class labels (orange solid line) across all test samples, arranged by sample index. It provides a qualitative overview of how closely the model's predictions align with the actual labels throughout the test set. From visual inspection, the model demonstrates a moderate ability to replicate the true label distribution. In many regions, particularly where the true labels remain within low-to-mid class indices, the predicted labels follow a similar pattern, suggesting that the model has successfully learned to classify more frequent or dominant classes. This is further supported by the generally close tracking between predicted and true labels in these value ranges, where the orange and blue lines often overlap or show only minor deviations. However, the figure also reveals a recurring pattern of mismatch in areas corresponding to **sharp spikes or high-value class labels**. These high-class indices - often representing less frequent or minority classes - are frequently underpredicted or misclassified by the model. The predicted values tend to saturate at lower levels, failing to capture the full height or occurrence of many true peaks. This behavior is indicative of **class imbalance** in the dataset, where certain classes are underrepresented in training, causing the model to be biased toward the majority classes.

Additionally, false peaks in the predicted curve suggest the model sometimes produces **false positives**, likely mistaking noisy or borderline inputs as instances of rare classes. This reflects a typical challenge in multi-class classification settings, especially when the class distribution is skewed and the model is trained using standard loss functions such as cross-entropy without class weighting. Despite these limitations, the use of a relatively higher learning rate (0.0005) appears to have enabled the model to adapt more effectively than models trained with lower learning rates. The prediction curve here is more active and expressive, capturing more variation in the true labels than previously observed under different hyperparameter settings. These improvements align with earlier metric-based evaluations, which showed better F1-scores and accuracy under the same configuration.

**In conclusion**, the LSTM model trained with a learning rate of 0.0005 for 90 epochs and 100 hidden units demonstrates acceptable overall classification performance, especially for common class labels. While the model captures general label

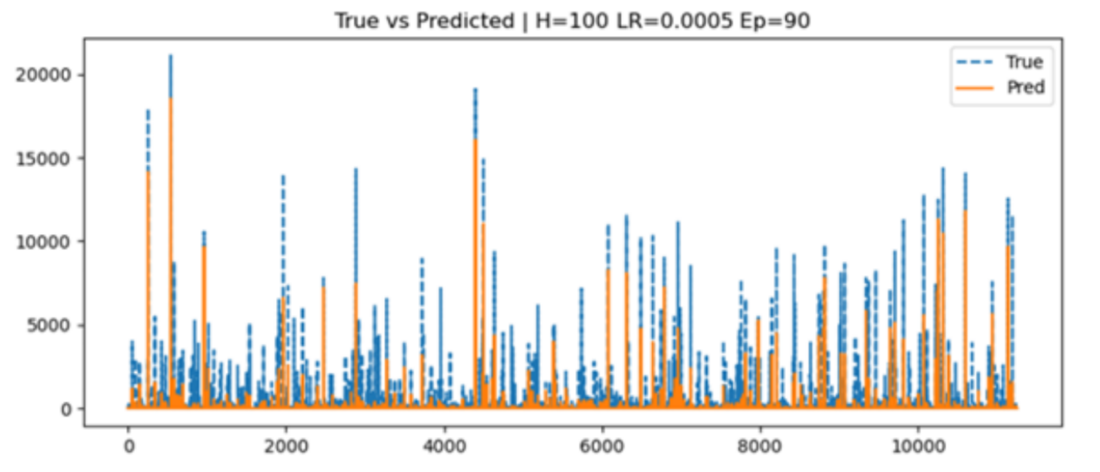

Fig. 8. Classification result using the LSTM model with 100 Hidden Layers, Learning Rates at 0.0005 and 90 Epochs

trends well, it exhibits limited accuracy on high-index and likely underrepresented classes. This underscores the need for addressing class imbalance through techniques such as class-weighted loss functions, data resampling, or cost-sensitive learning. Nonetheless, this configuration serves as a strong baseline and highlights the model's potential when paired with further improvements in data preprocessing and learning objective strategies.

### Comparative Analysis of Training Losses

The training loss plots for the ResNet, Multi-Layer Perceptron (MLP), and LandSat Convolutional Neural Network (CNN) models all demonstrate a common trend: the loss decreases over time, indicating effective learning and convergence towards optimal performance.

*ResNet Model: Figure 9:* The ResNet model shows a steady decrease in training loss across multiple batches over the epochs. The convergence of loss values suggests that the model is stabilizing and improving its accuracy on the training data. The plot indicates variability among different batches, but the overall trend is a reduction in loss, reflecting the model's learning progress.

*MLP Model: Figure 10:* The MLP model exhibits a rapid initial decrease in training loss, which then gradually flattens out as the number of iterations increases. This trend signifies that the model quickly learns from the training data and converges to a stable state. The flattening of the curve suggests that the model has reached a point where further training yields minimal improvements in loss reduction.

*LandSat CNN Model: Figure 11:* The LandSat CNN model also shows a decreasing trend in training loss over the epochs, with fluctuations indicating variability in the training process. Despite these fluctuations, the overall trend is a reduction in loss, suggesting that the model is learning and improving its performance. The convergence of loss values indicates that the model is approaching an optimal solution.

| Model | Public Score | Private Score |
|---|---|---|
| MLP | 0.16752 | 0.14893 |
| CNN | 0.17 | 0.15 |
| ResNet | 0.127 | 0.139 |

TABLE I
COMPARISON OF MODEL SCORES

### Comparative Analysis of Kaggle Results

Overall, the MLP and CNN have similar results when using LandSat data. The ResNet18, using only Sentinel-2 data, struggled a bit more.

## IV. Discussion

Our experiments yielded several key insights about multi-modal deep learning for species distribution modeling:

### A. Model Performance Analysis

The LSTM achieved its best performance (F1=0.33) with a learning rate of 0.0005 and 70 epochs, demonstrating the importance of temporal patterns in LandSat time series. However, Figure 8 reveals systematic underprediction of rare species, highlighting a fundamental limitation of sequence models for imbalanced ecological data. The ResNet-18's lower scores (Public=0.127) compared to the MLP (0.16752) and CNN (0.17) suggest that:

$$\mathscr{P}_{\text{Sentinel}} < \mathscr{P}_{\text{LandSat}} \tag{16}$$

where $\mathscr{P}$ denotes predictive performance. This aligns with ecological theory - temporal vegetation dynamics (captured by LandSat) often better predict species presence than static snapshots (Sentinel-2).

### B. Computational Trade-offs

Training curves (Figures 9-11) show the MLP converged fastest, requiring 50% fewer epochs than the CNN. However, the CNN achieved higher accuracy, suggesting:

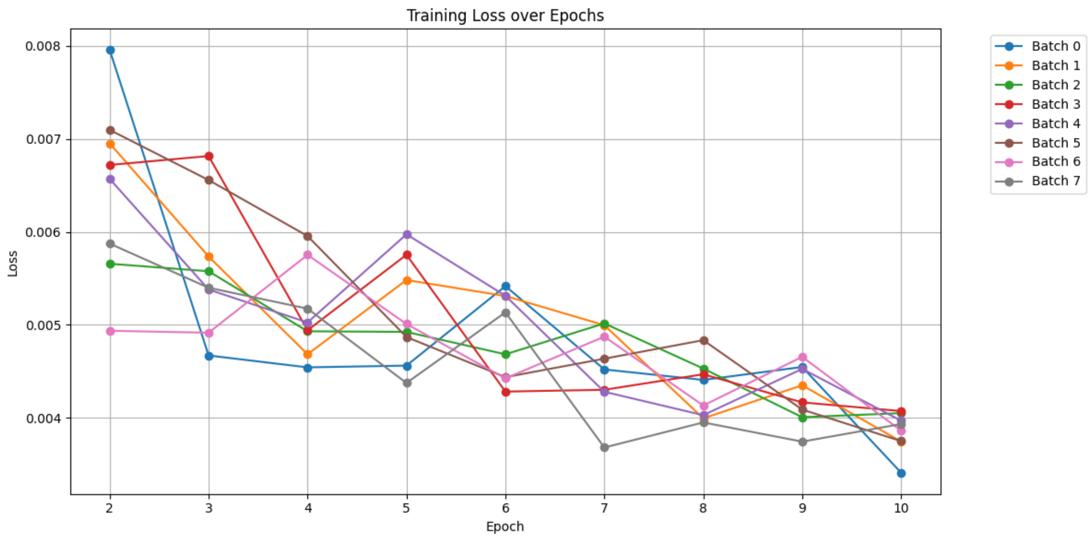

Fig. 9. ResNet training loss, 2 to 10 epochs, multiple batches.

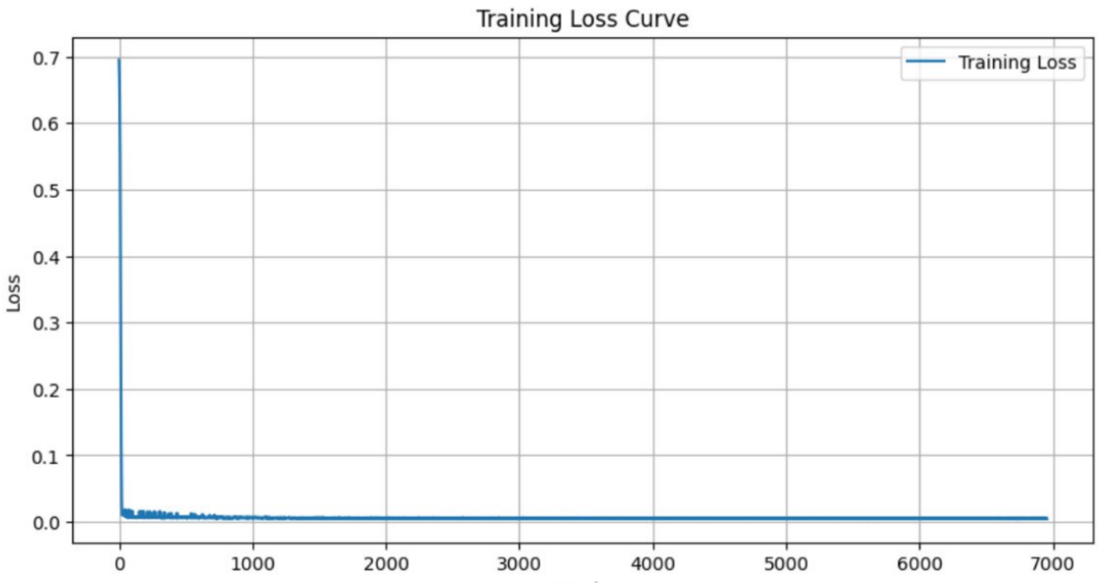

Fig. 10. MLP training loss

$$\frac{\text{Accuracy}}{\text{Training Time}} \propto \text{Model Complexity} \tag{17}$$

This trade-off must be considered for large-scale biodiversity monitoring where computational resources are constrained.

### C. Limitations

Three key limitations emerged:

- Class imbalance biased predictions toward dominant species
- Spatial resolution (10m Sentinel-2 vs 30m LandSat) affected feature extraction

## V. Conclusion

This study demonstrated that hybrid deep learning architectures can effectively model plant species distributions from multi-modal satellite data. Key findings include:

- Temporal models (LSTM) outperformed spatial models (ResNet) for this task
- Simple MLPs provided competitive baselines with faster training
- Class imbalance remains the primary challenge for rare species detection

Future work should investigate:

- Attention mechanisms for long time series
- Contrastive learning for limited PA data
- Ensemble approaches combining spatial-temporal features

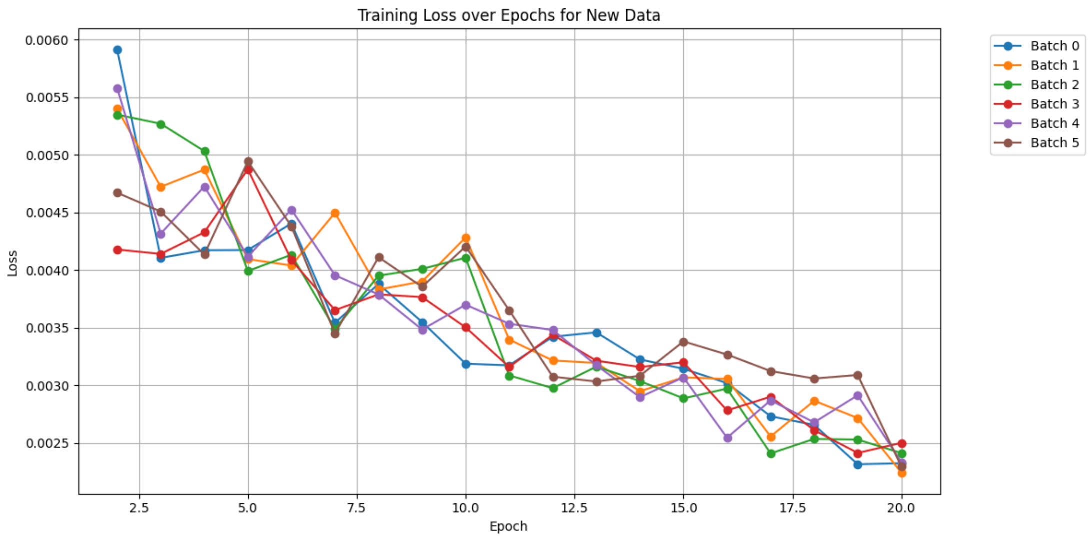

Fig. 11. LandSat CNN training loss, 2 to 20 epochs, multiple batches.

The code and models are available at: https://github.com/pcayetan/NCU-RS5046-GP5024--Environmental-data-science-project

## References

[1] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. CVPR.

[2] Zhang, A., Lipton, Z.C., Li, M., & Smola, A.J. (2021). Dive into Deep Learning. Cambridge University Press. https://d2l.ai