Species Prediction from Satellite Images and Environmental Data

Final Report
COMS 6998 E Advanced Topics Project Deep Learning

Kennedy Salamat, kas2319
Jannik Wiedenhaupt, jjw2196
Nathalie M. Hager, nmh2147.

***Abstract*—Maintaining biodiversity is critical for many aspects of society. This project aims to use deep learning techniques to streamline the process for monitoring biodiversity. We build several deep learning models – conventional convolutional neural network (CNNs), ResNets, and Transformers – in order to approach this problem. Ultimately, the best performing architecture on the validation set were the Transformers, followed by the ResNets, and then simple CNNs. It is noteworthy that the highest performing Transformers were those trained only on RGB satellite data, achieving a top-5 accuracy of 95% on the validation set. However, ResNets performed best on the test set, followed by simple CNNs, and then Transformers, suggesting bias in the validation set, and indicating that more research into the differences of vision Transformers and CNNs needs to be done. This project also became an exploration of working with a large, highly imbalanced dataset for deep learning, which we approached by thoroughly analyzing the dataset, implementing a custom data loading pipeline, and using various data augmentation techniques. The code can be found on Github at https://github.com/jannikjw/species-presence-prediction.**

I. Introduction and Motivation

Currently over 1 million species are at risk of becoming extinct [1]. Preserving biodiversity is critical for protecting the climate, reducing emissions, preventing disease outbreaks like the COVID-19 pandemic, preserving natural resources, informing public policy, and maintaining the economy [1] [2]. However, monitoring biodiversity is a challenging task, and currently relies in large part on "citizen scientists" who volunteer to make observations locally [2]. Thus, information on a global scale can be difficult to obtain in regions where there is a shortage of volunteers [2]. The ability to suggest which species are most likely to be present in a geographical location from easily obtainable satellite data would streamline the process of monitoring biodiversity. Such a model could lead to automatic species identification tools and support scientists and citizens in predicting existing species, by reducing the number of options they need to consider significantly [3]. The Kaggle GeoLifeCLEF 2022 competition challenges researchers to seek deep learning solutions to this problem [3].

In this paper, we investigate how different deep learning approaches – namely simple CNNs, ResNets, and Transformers, as studied in class – can be leveraged to identify the species present in a geographical region. The dataset provided by the Kaggle competition contained approximately 1.6 million observations regarding 17,000 species found in the United States and France [3]. Data included satellite images and tabular environmental data (e.g. soil, rainfall, temperature information) [3]. Due to the large size of the dataset, but relatively sparse number of observations per label, we had to implement custom data loading routines and several data preprocessing techniques, including a data augmentation system. Thus, in addition to comparing deep learning architectures discussed in class, this paper addresses the challenges inherent in working with large datasets.

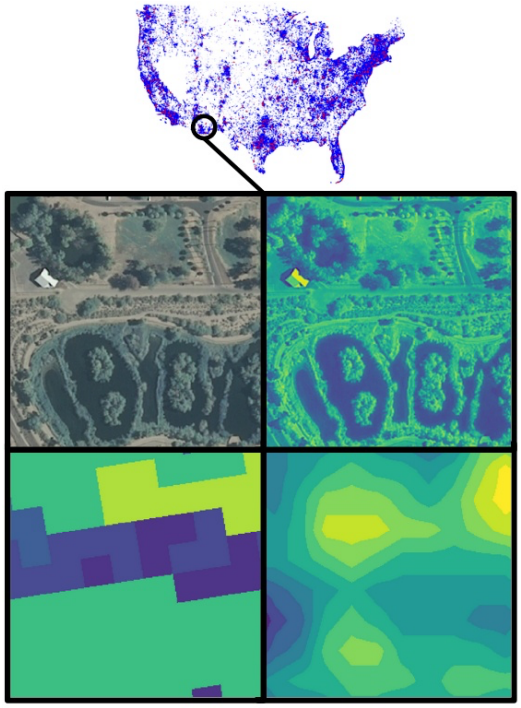


Fig. 1: Schematic provided by Kaggle competition showing satellite data. The dataset contains RGB image, near infrared, altitude, and land cover data.[3]

II. Related Work and Background

This project is set-up as a multi-class image classification task: Multiple species can occur in a given patch of land. Therefore, the goal is to generate a set of 5 candidate species that contains the correct label. Image classification has been discussed in class in connection to both CNN and vision Transformers and remains an active are of research. In this project we mainly used the knowledge obtained while taking this class and other deep learning classes at Columbia as well as research projects, when working on the computer vision learning part of this final project.

Deep learning approaches have been used by researchers in the past to try to count specific species present in a geographical location using satellite imagery (e.g. counting the number of elephants or whales) [4] [5]. However, our models do not have the same prior knowledge that a specific species is present, but instead have to successfully tackle the task of predicting the most common species from the underlying structure of the captured biomes.

ResNet are widely used for satellite image data classification, solving problems ranging from estimating crop yields to determining infrastructure quality. This is because ResNet's identity blocks combat the exploding/vanishing gradient problems that may arise with the CNN architecture [6]. Moreover, they allow the model to find the optimal number of layers to utilize, as re-training the model with varying numbers of layers can be slow and resource-intensive, especially when operating on a dataset of this scale. [7].

Recently, Transformer-based architectures have become increasingly popular as they promise to yield better results compared to state-of-the-art convolutional networks while requiring substantially fewer computational resources to train [8]. However, most research in vision transformers has been focused on clearly identifiable objects like ImageNet or CIFAR. These tasks serve as a foundation for our task, classifying satellite images. Recent research on satellite images using Transformer-based architectures include landcover classification [9] or for example deforestation identification [10].

Our tasks however, requires a more implicit classification. Deforestation classification for example classifies images into one of 17 classes which exhibit clearly identifiable patterns like clouds, smoke, burnt wood etc. [10]. Our task on the hand has less distinct labels. Each patch of land can match many different labels and the distinctions between habitable spaces for different species is of indistinguishable to the untrained eye.

However, Transformers are efficient at recognizing which aspects of any given data are most important to make a decision. Therefore, we examine whether Transformers' attention mechanism is better at extracting the underlying structures of the given satellite images than convolutional networks.

III. Data

A. Overview of Dataset

The dataset consists of 1.6 million observations, with each observation consisting of both image and numerical data. Per observation, we have four image patches: RGB, near infrared, landcover, and altitude (see Figure 1). In addition, for each observation, we have one environmental vector, which contains bioclimatic and pedological variables, as well as two gps-coordinates. There are 17k species (= labels) in the entire dataset, and thus, on average, we have ca. 94 observations per species.

B. Loading the Dataset

Compressed on kaggle, the dataset has a size of about 60GB. However, when loaded into RAM as a python list, the RGB patches alone are 330GB large (even larger if formatted as a numpy array, which is needed for training). Since not even high-ram machines (we tried machines with up to ca. 600 GB of RAM) on the Google Cloud Platform could fit the entire dataset at once into memory, we were not able train on the entire dataset from memory. Instead, we developed a custom data generator in tensorflow to read the dataset from disk as we train, batch by batch.

We developed one custom patch generator which loads the rgb data for each observation for our rgb-only models. In addition, in order to use both patch and environmental vector data as input to the multi-modal models, we wrote a second custom generator. This generator would load patches associated with the inputted observation ids, and concatenate RGB, Near IR, Altitude IR, and Landcover data into a $(256, 256, 6)$ tensor. This generator would also load tabular environmental vector data, which was comprised of bioclimatic and pedologic information. For each observation, this has the shape of a $(27,)$ tensor. However, for the chosen labels, some of the numeric data is missing. To not reduce the observations per label further, we use the SimpleImputer from the Scikit-Learn Python library to fill in missing values [11].

We used the following techniques to speed up the data loading and processing procedure for both data generators:

- **Pre-fetch batches.** Loading each batch sequentially means to load a batch via CPU, train on the batch via GPU, and then load another batch via CPU, and so on. This is not efficient, since while the CPU loads a new batch, the GPU stays idle, and while the GPU trains on the batch, the CPU stays idle. Instead, we ideally want both of them busy at all times, and thus we prefetch the batches. This means that while the GPU is training, the CPU continues to load the next upcoming batches, so that ideally, there are always new batches for the GPU to train on, thus (ideally) removing the CPU as a bottleneck for training.
- **Use multiple CPU cores in parallel when reading data from disk.** Since one CPU alone was too slow to keep the GPU consistently busy with new batches, we use multiple CPU cores in parallel to prefetch the batches. We used maximum *CPU cores * vCPUs per core* threads in parallel (e.g. 12 CPU cores with 2 vCPUs per core make a maximum of 24 threads running at the same time) for prefetching.

- **Data parallelism (synchronous, all-reduce) during training by using multiple GPUs.** We used multiple GPUs for training by employing the concept of data parallism. We used synchronous, all-reduce data parallelism as this was conveniently implemented in the tensorflow API [12], while potentially faster techniques such as asynchronous data parallelism would have required a time consuming implementation effort from our side. However, for future work on this project, it could be interesting to implement asynchronous data parallelism for a further performance speedup.

We each experimented with different machines until we each found a setting that worked well for training our specific models. For example, for a small CNN (see section IV-A1), all these strategies together resulted in a 60% speedup of time/epoch on a high-cpu machine (96 cores) with 4 Nvidia T4 GPUs.

C. Dataset Imbalance

We analyzed the dataset and realized that it is highly imbalanced. While on average, there are ca. 94 observations per label, the amount of observations per label actually greatly varies within the dataset. This is shown in Figure 2.

Due to this imbalance as well as the dataset as a whole being too large to efficiently train on anyways, even with our optimized load-from-disk routine (see section III-B), we decided to focus on a subset of all labels (species) within the dataset, instead of looking at the entire dataset.

That is, we only focused on all labels that have between 2000 to 3000 observations per label. This yielded a dataset significantly smaller than the real dataset, but too large to load into memory. 10 percent of the observations associated with each of these labels were randomly selected to comprise the test set, while the remaining 90 percent of the observations were taken as the training set. A random seed was used to ensure that all of the models were trained and tested on the same data, to allow comparisons between models. The validation set was selected from the validation subset provided by the Kaggle competition, as we took all observation ids from the validation subset that matched the 31 training/test labels. Ultimately, the size of our training, validation, and test sets were $63,278$, $1,831$, and $7,016$ observations, respectively.

D. Data Augmentation

We use augmentation to improve the generalization of our model. As satellite images can have different meanings based on their orientation, coloring, and orientation, augmentations need to be selected carefully according to Ghaffar et al. [13]. Therefore, we apply random rotations at a small factor of 0.02, random horizontal flipping, and random contrast to emphasize light differences. Some of these augmentations can be seen in Figure 3.

IV. Implementation

A. Overview of Models

As the goal of the project was to examine the training and performance differences of traditional convolutional neural networks and Transformers, we implemented two CNNs from scratch to exploit the specific domain as much as possible as well as a trained from scratch ResNet [14], and a pre-trained ViT [8] to take advantage of recent advances in image processing.

1) Small CNN: We developed a small convolutional neural network to obtain a relatively simple baseline to compare the other, more complex models against. There are two versions of this model:

- CNN-S RGB-Only: This network consists of three convolutional layers, followed by two dense layers and one output layer. It takes only rgb-image patches as the input.
- CNN-S Multi-Modal: For each observation, this network takes as input all patches as well as the numerical environmental vector data. As shown in Figure 4, this

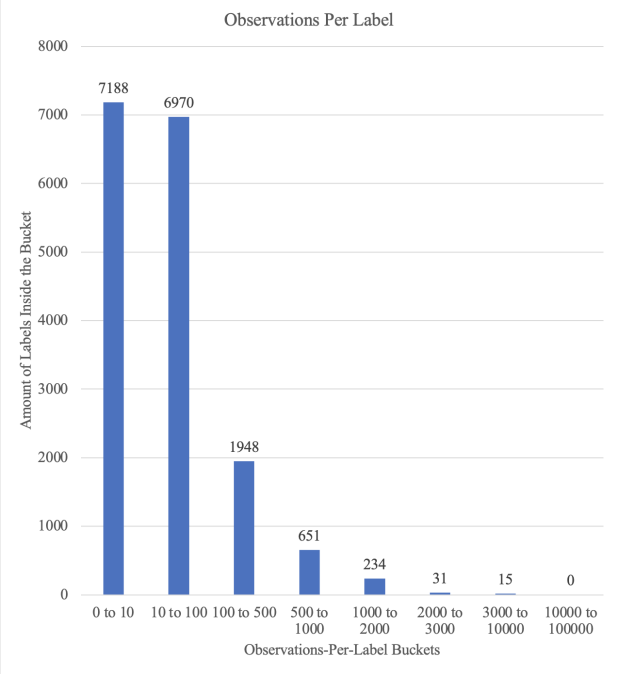


Fig. 2: Analysis of observations per label within the dataset. "0 to 10" bucket means that for all labels in this bucket, there are 0 to 10 observations per label.

Fig. 3: Applied data augmentation. We added random contrast (factor$=0.1$), random horizontal flipping, and random rotation (factor$=0.02$).

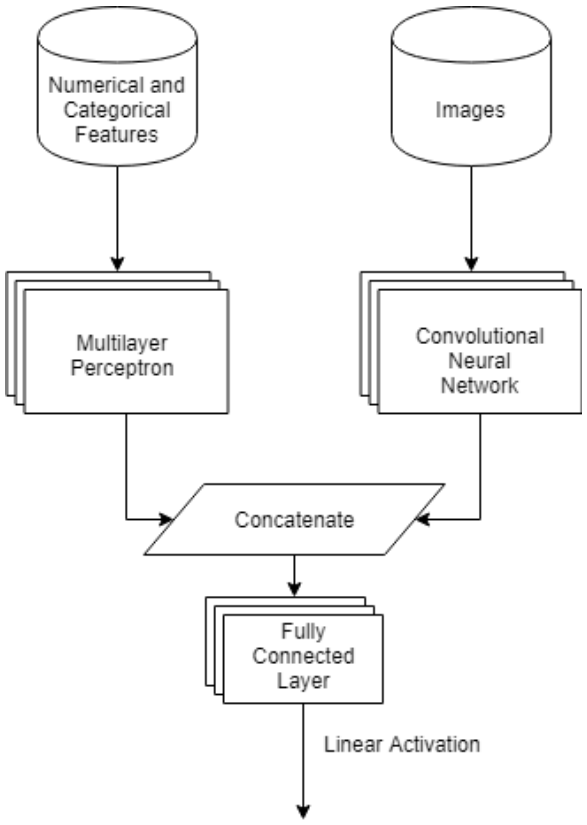


Fig. 4: Conceptual overview of "Small CNN Multimodal".[15]

model consists of a convolutional part which trains on the image data and of a fully-connected part (an MLP) which trains on the numerical data. The outputs of both parts are then concatenated and passed through a final fully connected layer and an output layer. This setup allows the model to train on different kinds of input data at once.

A detailed model summary of both versions can be found in the submitted code.

2) Large CNN: This model matches the ResNet model (see IV-A3) in number of parameters (both have around 2.3 million). In contrast to the ResNet model, the large CNN model is a conventional CNN in that it doesn't consist of any specifically-designed underlying structures such as ResNet's identity blocks. Instead, it has 8 convolutional layers, followed by 3 dense layers and an output layer.

We designed this model so that we are able to experimentally evaluate whether the specific underlying structure of a ResNet results in a better performance on our dataset, compared to a conventional, approximately equally large, CNN. Just as the small CNN IV-A1 and the ResNet IV-A3, this model exists in a rgb-input-only and multi-modal-input version.

We spent a significant amount of effort tuning the large CNN to see how far we can push the performance of a conventional CNN, using the following strategies:

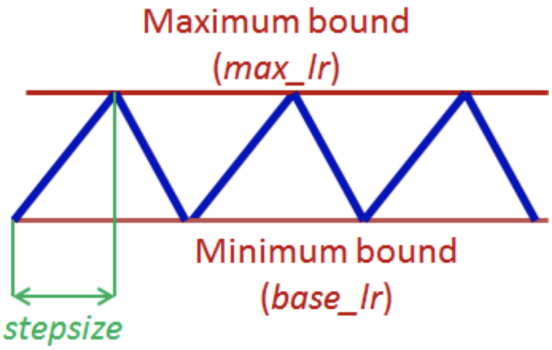


Fig. 5: Cyclic learning rate as training progresses. Image source: [16]

- Cyclic Learning Rate: The idea behind cyclic learning rates is, "instead of monotonically decreasing the learning rate (...) [, let] the learning rate cyclically vary between reasonable boundary values." [16]. This is shown in Figure 5. During training, this allows the model to both slowly progress towards the absolute minimum (without ping-ponging around the minimum due to too large learning rates) as well as overcome a local minimum (without being stuck in it due to too small learning rates). We used the method presented in [16] to determine the best maximum and minimum learning rate bounds experimentally, as shown in Figure 6.
- Weight Initialization: To prevent exploding and/or vanishing gradients, we initialized the weights of the large CNNs via the *He Initialization Scheme* [17], which is recommended for networks using ReLu activation functions. [18]
- Regularization: We used Dropout layers for both the convolutional and the dense layers with a probability of 0.1, which we determined via manual experimental tuning.
- Pooling: The large CNNs also contain pooling layers after most convolutional layers. We have found that max-pooling layers work better than average-pooling layers for our dataset. This makes sense as the max-pooling layers maintain the brighter spots in the patches, thereby maintaining potentially unique features in a satellite patch (e.g. when looking at the altitude lines in the altitude patch).

3) ResNet: Initially, we planned to use a pre-trained ResNet-50. We built a pre-trained, multimodal ResNet-50, freezing most of the layers, and adding on a convolutional layer and a few dense layers at the end. However, due to the large size of the dataset, the model took approximately 4 hours per epoch, and thus, we had to try another approach due to computational constraints.

We constructed a ResNet from scratch, to provide us with the flexibility to customize the architecture to mirror that of the simple CNN so we could draw direct comparisons, as well as have more flexibility with the multimodal data input pipeline

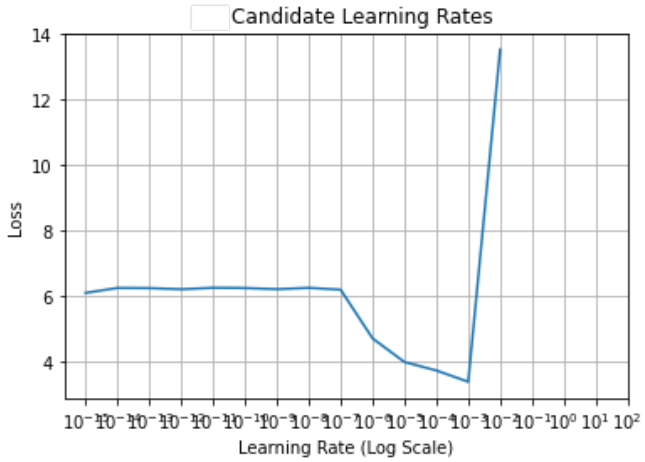


Fig. 6: Analysis of candidate learning rates for our large CNN. The optimal min bound is at the start of the decrease (e1-7), while the optimal max bound is at the end of the decrease (1e-3), as per the method presented in [16].

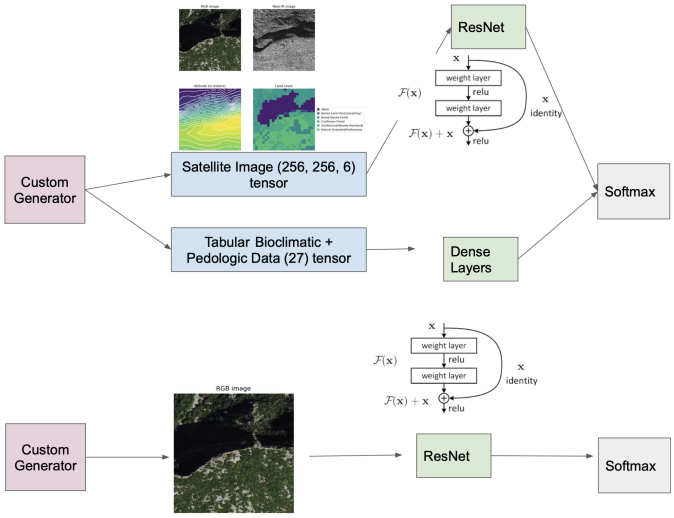


Fig. 7: ResNet architecture overview for RGB and multimodal data input pipelines.

and be able to design a smaller custom ResNet for the sake of computational time [19] [20].

ResNets were trained on two types of data: one was trained on RGB-only and the other on multimodal data comprised of environmental tabular data as well as all four types of satellite images. The ResNet that used multimodal data took as input the satellite data as a $(256, 256, 6)$ tensor and the tabular environmental vector data as a (27) tensor. The satellite data was processed through a series of convolutional network and identity blocks , which included Conv2D layers, Batch Normalization, and ReLU Activations [19] [20]. The environmental data was feed through a few dense layers with ReLU Activation. The two tensors were then concatenated and put through a softmax layer to obtain a probability distribution across the 30 possible species id labels.

In order to combat overfitting observed by the model, techniques tried included:

- Varying the learning rate
- Utilizing data augmentation
- Weight decay in the form of L2 Bias and Weight regularizers [21]
- Reducing complexity of the conv-net block to just 1 Conv2D per block
- Reducing number and size of dense layers to decrease the number of trainable parameters
- Introducing dropout [22]
- Reducing the number of identity blocks
- Early Stopping with respect to val accuracy

A total of four of the ResNet variations tested are discussed later in this paper. Many more variations were tested as well, with hyperparameter fine-tuning, but for the purposes of the report, we focus on models implementing certain overfitting reduction techniques mentioned above:

- (1) Larger Conv-net blocks + Dropout, RGB-Only
- (2) Larger Conv-net blocks + Dropout, Tabular Data + All Satellite Image Types
- (3) Smaller Conv-net blocks, Tabular Data + All
- (4) Smaller Conv-net blocks + L2 Regularization, Tabular Data + All Satellite Image Types

4) Pre-trained ViT: We train three configurations of ViT. First, we acquire are baseline by fine-tuning the original ViT, pre-trained on ImageNet-21k, on RGB images only. Second, we use a version of ViT that was fine-tuned on a satellite image dataset on RGB images only. Third, we fine-tune the satellite image ViT on all the given satellite data.

We use ViT [8] trained on ImageNet-21k as a base, as provided by Huggingface [23]. This model is shown in Figure 8. The idea behind this model, ViT-IN-RGB, is to establish a baseline for image classification and exploit the object recognition features that the pre-trained ViT comes with. Considering however that ImageNet is very different from the GeoLifeCLEF dataset, we also fine-tune two ViT models, ViT-ES-RGB and ViT-ES-Sat, that were trained on the EuroSAT dataset [24], a dataset that includes $27,000$ RGB satellite images.

Thus, we use the pre-trained ViT and fine-tune on the our dataset and classification task. The base ViT architecture expects inputs in the shape (C, H, W), where (H, W) is the resolution of the input image, C is the number of input channels. As the normal RGB format is (H, W, C), swap the axes of each image and resize it to match the size of the images in the ImageNet-21k dataset (224×224). We further augment the data to improve the models ability to generalize and mitigate the low number of observations per label.

The ViT model then converts the augmented images into a sequence of flattened 2D patches of the dimensions, which are then fed to the pre-trained tokenizer.

In addition to the RGB-based ViT models, we also fine-tuned the ViT that was pre-trained on EuroSAT on all of the available satellite patches, i.e. RGB, near infrared, landcover, and altitude information. For this model, ViT-ES-Sat, we concatenate the images along the C axis.

For the classification task at hand, we add a multi-layer perceptron with dropout and a classification layer to the head of the ViT model. Finally, the model is fine-tuned on the flattened image patches using categorical cross entropy as specified in section V.

```
Model: "vit-rgb-pretrained-imagenet"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rgb_input (InputLayer)      [(None, 256, 256, 3)]     0

 data_augmentation (Sequenti  (None, 3, 224, 224)      0
 al)

 tf_vi_t_model (TFViTModel)  TFBaseModelOutputWithPoo  43862016
                             ling(last_hidden_state=(
                             None, 197, 768),
                              pooler_output=(None, 76
                             8),
                              hidden_states=None, att
                             entions=None)

 tf.__operators__.getitem (S  (None, 768)              0
 licingOpLambda)

 dense_17 (Dense)            (None, 2048)              1574912

 dropout_34 (Dropout)        (None, 2048)              0

 dense_18 (Dense)            (None, 1024)              2098176

 dropout_35 (Dropout)        (None, 1024)              0

 dense_19 (Dense)            (None, 30)                30750

=================================================================
Total params: 47,565,854
Trainable params: 47,565,854
Non-trainable params: 0
_________________________________________________________________
```

Fig. 8: Examplary architecture for the pre-trained ViT models. Shown above for the baseline model, pre-trained on ImageNet-21k.

5) Multimodal ViT: Since one of the goals of this paper is to develop efficient multimodal models for large datasets, we developed a multimodal version of ViT, called TabViT. The model takes advantage of both the environmental data and the RGB images in the dataset. This approach aims to exploit the attention mechanism of Transformers in which they can evaluate the combined the inputs continuously, whereas CNN-based architectures need different branches for image and numeric data.

An overview of the model is depicted in Figure 9. The standard Transformer receives as input a 1D sequence of token embeddings. To handle 2D images, we reshape each image $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ into a sequence of flattened 2D patches $\mathbf{x_p} \in \mathbb{R}^{N \times (P^2 \cdot C)}$, (P, P) is the resolution of each image patch and $N = HW/P^2$ is the resulting number of patches. The Transformer uses constant latent vector size D through all of its layers, so we flatten the patches and map to D dimensions with a trainable linear projection. This method is equal to the image processing in the orginal ViT paper by Dosovitskiy et. al. [8].

As introduced earlier, the dataset contains environmental, pedologic, and location data. We concatenate all the given data based on their respective observations. Therefore, the input length I per label is sum of the number of patches N and the number of cells T in the table, $I = N+T$. As with the images each table cell v is projected into a latent vector of size D.

Position embeddings are added to these token embeddings to retain positional information. We use standard learnable 1D position embeddings for image patches and cells where each position is recorded in reference to either the image or the table, respectively. Lastly, we add segment embeddings to differentiate between image and tabular data (Eq. 1).

The Transformer encoder [25] consists of alternating layers of multi-headed self-attention and MLP blocks (Eq. 2, 3). Layer normalization (LN) is applied before every block, and residual connections after every block [26], [27] to replicate the ViT structure [8].

$$\begin{aligned} \mathbf{z}_0 = &\left[\mathbf{x}_p^1\mathbf{E}; \mathbf{x}_p^2\mathbf{E}; \cdots; \mathbf{x}_p^N\mathbf{E}; \mathbf{v}^1\mathbf{E}; \mathbf{v}^2\mathbf{E}; \cdots; \mathbf{v}^T\mathbf{E}\right] \\ &+ \mathbf{E}_{pos} + \mathbf{E}_{seg}, \\ &\mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}, \mathbf{E}_{pos} \in \mathbb{R}^{(N+1) \times D}, \\ &\mathbf{E}_{seg} \in \mathbb{R}^{(2) \times D} \end{aligned} \tag{1}$$

$$\mathbf{z}'_\ell = \mathrm{MSA}\left(\mathrm{LN}\left(\mathbf{z}_{\ell-1}\right)\right) + \mathbf{z}_{\ell-1}, \quad \ell = 1 \ldots L \tag{2}$$

$$\mathbf{z}_\ell = \mathrm{MLP}\left(\mathrm{LN}\left(\mathbf{z}'_\ell\right)\right) + \mathbf{z}'_\ell, \quad \ell = 1 \ldots L \tag{3}$$

This papers covers two multimodal Transformer architectures. The first one, MM-ViT-RGB, is trained on the tabular data and the RGB images and therefore has $C = 3$. The second one, MM-ViT-Sat, is trained on the tabular data and on all four types of satellite images and therefore has $C = 6$.

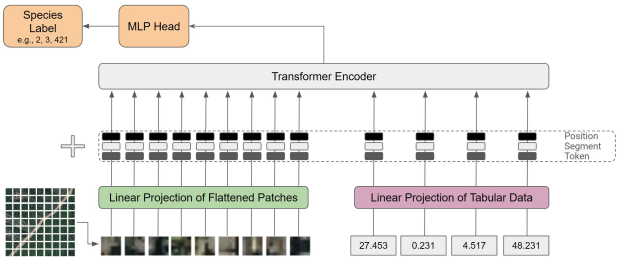


Fig. 9: Image patches are combined with numeric cell values. Their linear projections are added to segment and position embeddings. The two inputs, combined with a Transformer encoder allow the self-attention to discover alignments between the environmental data and the images. The architecture is trained with a multi-layer perceptron on the species classification task.

B. Implementation Details

The models described in the previous section have been implemented in a Jupyter Notebook [28] in Python 3 [29] using the Tensorflow Keras Deep Learning Framework [30] and its documentation [31]. The code can be found on Github at https://github.com/jannikjw/species-presence-prediction (please refer to all branches for the full code). The models have been implemented and trained on the Google Cloud Platform (GCP) [32].

V. Training

All models have been trained for a maximum of 50 epochs. We only trained using 30 labels and all their respective observations because the high-resolution images would otherwise require extended training times and computing resources that were not available for this project.

All specified models use the sparse categorical cross entropy loss function which uses the categorical cross entropy loss

$$\text{Loss} = -\sum_{i=1}^{n} y_i \cdot \log \hat{y}_i$$

but handles labels that are not one-hot encoded. The used optimizer varies from model to model to achieve the best performance in the shortest training time. Generally, we use the Adam [33] and AdamW [34] optimizer depending on which yielded better performance as they have been identified as most effective in similar research [8], [35]. [36].

We used the following callback functions from the TensorFlow library.

1) tf.keras.callbacks.ModelCheckpoint
   This function saves the best model encountered during training so far to file. This way, we only need to load the model after training has stopped in order to obtain the best model (here, that means the model with the highest validation accuracy) seen during training.
2) tf.keras.callbacks.EarlyStopping
   Stops training early in case the validation loss has not been increasing during the past 10 epochs. Using this callback allows us to pass in a high value for max epochs (of 50), which essentially achieves the effect of stopping training as soon as the validation accuracy does not improve anymore. This prevents over-fitting, while the initially high max epoch number ensures that we do not stop training too early.
3) tf.keras.callbacks.CSVLogger
   Logs accuracy and loss on both the training and the validation set to a file after every epoch. We used this data to plot the graphs in section VI.

VI. Analysis and Results

A. Validation Set

The performance on the validation set of all models during training is shown in Figure 10. We can clearly see that among the three types of architectures, Transformers had the highest performance on the validation set by far, followed by ResNets, and then by the conventional CNNs (both small and large).

It is intuitive that the small CNN performs by far the worst out of all models, as its simple structure and small amount of learnable parameters likely do not allow it to fully learn to understand the subtle differences between the species present on the satellite data. The large CNNs show a clear improvement over the small CNNs in terms of performance, which shows that a larger amount of learnable parameters is needed to perform well on this dataset. For all CNNs, the multimodal ones outperformed the RGB-only ones. This clearly shows the value of the additional patches and environmental vector data in addition to plain RGB satellite pictures.

We observe that the ResNets outperform the conventional CNNs. While the best-performing CNN (MM-CNN-L) manages to reach a validation accuracy similar to some of the Resnets, it is far from outperforming the best performing ResNet. This is especially noteworthy since the large CNN was specifically tuned via He weight initialization to avoid the vanishing gradient problem. ResNets strive to avoid vanishing gradients through the introduction of their identity mappings. Our results show that even a carefully weight-initialized CNN of almost identical size to a ResNet does not perform as well, and thus we can clearly see that the purposeful (albeit more demanding in terms of implementation) design of ResNets pays off.

Not only do the Transformer-based architectures outperform the CNNs and ResNets on the validation set, but they also reach the same accuracy on the validation set much faster than CNNs and ResNets. In the following paragraphs, we are analyzing the performance of the Transformer-based architectures with respect to each other.

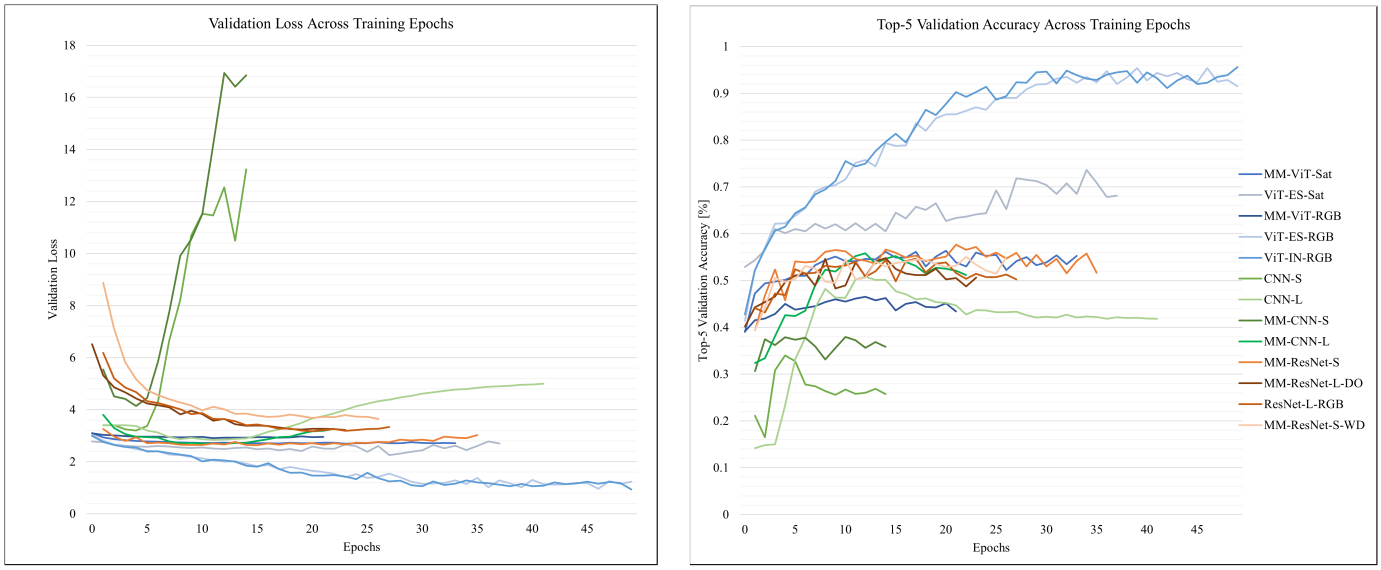
From Figure 10, we can clearly see that ViT-IN-RGB by far outperforms all other models. It achieves highest overall validation accuracy, 73%. It is closely followed by another Transformer-based model, ViT-ES-RGB (72%). After 50 epochs both these models also reach a top-5 validation accuracy of over 95%.

The third pre-trained model, ViT-ES-Sat, does not perform as well. Its peak validation accuracy is 37%. This is likely due to the fact that ViT was pre-trained on 3 channels and does not generalize well to additional channels, particularly with different structures - RGB is very different from altitude data.

Furthermore, the performance of the pre-trained models supports the hypothesis that Transformers need large amounts of data to achieve good performance. This also indicates their potential to be applied to the full dataset of observations.

Multimodal models: In Figure 10, we can also observe that the multimodal ViTs abort training early, indicating convergence without fully exploring the feature space. This might be due to the lower number of parameters as the model is not able to learn the full representation of the the data like the pre-trained models.

While the performance of the pre-trained ViT models is satisfactory, the multimodal models, that have not been pre-trained only achieve a top-5 validation accuracy of 43% to



(a) Validation loss.

(b) Top-5 validation accuracy.

Fig. 10: Model performance during the training process. Training was stopped early when the validation accuracy did not improve for 10 epochs. Legend applies to both graphs.

55%. However, the comparison of the model using RGB images and the environmental data and the model that uses all satellite patches shows that the additional data improves the predictive abilities of the model. It also indicates that Transformer can adapt to different inputs given an appropriate encoding.

Additionally, when observing the performance of the different models, one must not forget that the multi-modal models have about one third of the parameters of the pre-trained models (17m compared to 47m). While it cannot be said for certain that more parameters would improve performance to the same level as the one of the pre-trained ViT models, there is little indication that it would not. The smaller size of the multi-modal models is due to the smaller projection dimension (64 compared to 768).

B. Test Set

The test set results are shown in Figure 11. Surprisingly, Transformers performed the worst on the test-set, with ResNets having the highest performance followed closely behind by simple CNNs. This possibly has to do with the size and data present in the validation set. As the validation set is under 2,000 images, and is much smaller than both the training set, and test set, it's possible that the data present in the validation set is not a representative sample of the data and has some form of bias. The small size of the validation set would also explain the fluctuations in validation accuracy observed for ResNet.

For the CNNs, we observe that the multi-modal models are outperforming their respective rgb-only counterparts, just as on the validation dataset. Similarly to the validation set, the CNNs perform worse than the ResNets, thus further showing the superiority of ResNet's identity-block based design over a conventional CNN design (such as CNN-S and CNN-L).

From the overfitting reduction techniques tried for ResNet (decreasing the number of Conv2D layers in the conv-net blocks, introducing weight decay in the form of L2 weight/bias regularization, and adding dropout), none of them seemed to have much of an effect on the test accuracies.

VII. Conclusion and Future Work

Maintaining biodiversity is critical for many aspects of society. In this project, we have employed multiple deep learning techniques to streamline the process for monitoring biodiversity. This project also became an exploration of working with a large, highly imbalanced dataset for deep learning, which we approached by thoroughly analyzing the imbalance of the dataset and deciding to focus on a specific part only (to gain a somewhat balanced dataset), implementing a custom data loading pipeline, and using various data augmentation techniques.

We built several deep learning models (conventional CNNs, ResNets, and Transformers) in order to approach this problem. Ultimately, the best performing architecture on the validation set were the Transfomers, followed by the ResNets, and then conventional CNNs. This result shows the superior abilities of Transformers. It also shows that the specific design structure of ResNets outperforms even highly-tuned, equally-large (in terms of parameters) CNNs.

In contrast to the validation set, ResNets performed best on the test set, followed by conventional CNNs, and then Transformers, suggesting bias in the validation set. Interestingly, the highest performing Transformers were those trained only on RGB satellite data, achieving a top-5 accuracy of 95%.

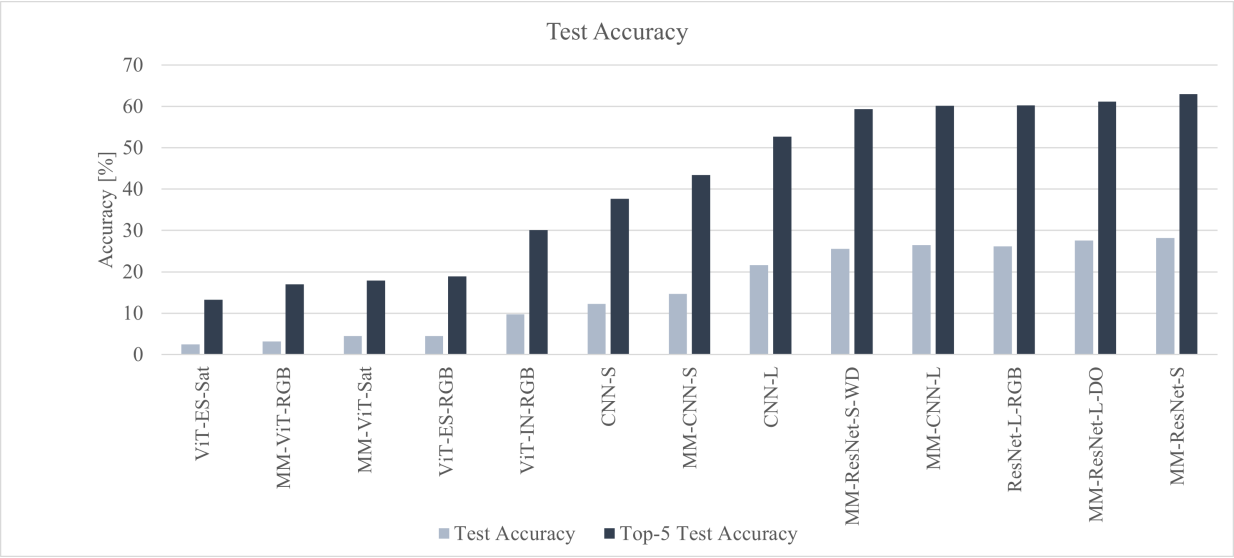


Fig. 11: Performance of each model on the test dataset.

A future direction for continuing this project would be to explore how we could train with the full dataset, even with the highly imbalanced parts of it. It could also be interesting to run Multimodal nets with more parameters (this was limited for us by computing resources), and more in-depth hyperparameter tuning for the ResNets and Transformers, similarily to what we have done for the large CNNs.

Another interesting direction could be to explore the use of GPS coordinates in our multimodal models. Researchers have reported that GPS coordinates are too granular to be used in their raw form. Thus, one technique that's become increasingly common is to divide the geographical region of interest into grids and use one-hot encoding to represent which region the GPS coordinates correspond to [37]. Recently, this embedding has been optimized in a model entitled "GPS2Vec" [37]. In the future, we would like to try to implement this embedding system to allow our models to properly utilize GPS data.

REFERENCES

[1] J. Shaw, "Why is biodiversity important?" *Conservation*, 2018. [Online]. Available: https://www.conservation.org/blog/why-is-biodiversity-important

[2] M. J.O.Pocock, M. Chandler, R. Bonney, I. Thornhill, A. Albin, T. August, S. Bachman, P. M. Brown, D. Gasparini, F. Cunha, A. Grez, C. Jackson, M. Peters, N. R. Rabarijaon, H. E.Roy, TaniaZaviezo, and FinnDanielsen, "Chapter 6 - a vision for global biodiversity monitoring with citizen science," *Advances in Ecological Research*, vol. 59, pp. 169–223, 2018. [Online]. Available: https://doi.org/10.1016/bs.aecr.2018.06.003

[3] "Geolifeclef 2022 - lifeclef 2022 x fgvc9," *Kaggle*, 2022. [Online]. Available: https://www.kaggle.com/c/geolifeclef-2022-lifeclef-2022-fgvc9

[4]

[5]

[6] G. Philipp, J. Carbonell, and D. Song.

[7] S. Goodman, A. BenYishay, and D. Runfola.

[8] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," *CoRR*, vol. abs/2010.11929, 2020. [Online]. Available: https://arxiv.org/abs/2010.11929

[9] A. J. Stewart, C. Robinson, I. A. Corley, A. Ortiz, J. M. L. Ferres, and A. Banerjee, "Torchgeo: deep learning with geospatial data," 2022. [Online]. Available: https://openreview.net/forum?id=ZgV2C9NKk6Q

[10] M. Kaselimi, A. Voulodimos, I. Daskalopoulos, N. Doulamis, and A. Doulamis, "Forestvit: A vision transformer network for convolution-free multi-label image classification in deforestation analysis," in *ICML 2021 Workshop on Tackling Climate Change with Machine Learning*, 2021. [Online]. Available: https://www.climatechange.ai/papers/icml2021/40

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[12] Google. (2022) tf.distribute.mirroredstrategy. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/distribute/MirroredStrategy

[13] M. Ghaffar, A. Mckinstry, T. Maul, and T.-T. Vu, "Data augmentation approaches for satellite image super-resolution," *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. IV-2/W7, pp. 47–54, 09 2019.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[15] L. Ghilardi. (2019) Combining images and numerical values in a deep neural network. [Online]. Available: https://www.mathworks.com/matlabcentral/answers/469940-combining-images-and-numerical-values-in-a-deep-neural-network

[16] L. N. Smith, "Cyclical learning rates for training neural networks," 2017.

[17] H. Kaiming, Z. Xiangyu, R. Shaoqing, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," 2015.

[18] D. Godoy. (2018) Hyper-parameters in action! part ii — weight initializers. [Online]. Available: https://towardsdatascience.com/hyper-parameters-in-action-part-ii-weight-initializers-35aee1a28404

[19] S. Bhattacharyya, "Understand and implement resnet-50 with tensorflow 2.0," *Towards Data Science*, 2020.

[20] yasho_191, "How to code your resnet from scratch in tensorflow?" *Analytics Vidhya*, 2021. [Online]. Available: \url{https://www.analyticsvidhya.com/blog/2021/08/how-to-code-your-resnet-from-scratch-in-tensorflow/}

[21] J. Brownlee, "How to use weight decay to reduce overfitting of neural network in keras," *Machine Learning Mastery*, 2018. [Online]. Available: \url{https://machinelearningmastery.com/how-to-reduce-overfitting-in-deep-learning-with-weight-regularization/}

[22] ——, "Dropout regularization in deep learning models with keras," *Machine Learning Mastery*, 2020. [Online]. Available: https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/
[23] Google. (2022) Huggingface: google/vit-base-patch16-224-in21k. [Online]. Available: https://huggingface.co/google/vit-base-patch16-224-in21k
[24] P. Helber, B. Bischke, A. Dengel, and D. Borth, "Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2019.
[25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762
[26] Q. Wang, B. Li, T. Xiao, J. Zhu, C. Li, D. F. Wong, and L. S. Chao, "Learning deep transformer models for machine translation," *CoRR*, vol. abs/1906.01787, 2019. [Online]. Available: http://arxiv.org/abs/1906.01787
[27] A. Baevski and M. Auli, "Adaptive input representations for neural language modeling," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=ByxZX20qFQ
[28] J. Notebooks. (2022) Jupyter notebooks documentation. [Online]. Available: https://docs.jupyter.org/
[29] Python. (2022) Python - official website. [Online]. Available: https://www.python.org/
[30] Google. (2022) Tensorflow - official website. [Online]. Available: https://www.tensorflow.org/
[31] ——. (2022) Tensorflow - api documentation. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf
[32] ——. (2022) Google cloud services. [Online]. Available: https://cloud.google.com/
[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014. [Online]. Available: https://arxiv.org/abs/1412.6980
[34] I. Loshchilov and F. Hutter, "Fixing weight decay regularization in adam," *CoRR*, vol. abs/1711.05101, 2017. [Online]. Available: http://arxiv.org/abs/1711.05101
[35] L. Yuan, Y. Chen, T. Wang, W. Yu, Y. Shi, F. E. H. Tay, J. Feng, and S. Yan, "Tokens-to-token vit: Training vision transformers from scratch on imagenet," *CoRR*, vol. abs/2101.11986, 2021. [Online]. Available: https://arxiv.org/abs/2101.11986
[36] S. P. Mohanty, J. Czakon, K. A. Kaczmarek, A. Pyskir, P. Tarasiewicz, S. Kunwar, J. Rohrbach, D. Luo, M. Prasad, S. Fleer, J. P. Göpfert, A. Tandon, G. Mollard, N. Rayaprolu, M. Salathe, and M. Schilling, "Deep learning for understanding satellite imagery: An experimental survey," *Frontiers in Artificial Intelligence*, vol. 3, 2020. [Online]. Available: https://www.frontiersin.org/article/10.3389/frai.2020.534696
[37] Y. Yin, Y. Zhang, Z. Liu, S. Wang, R. R. Shah, and R. Zimmermann, "Gps2vec: Towards generating worldwide gps embeddings," *IEEE Transactions on Multimedia*, vol. 24, pp. 890–903, 2021. [Online]. Available: 10.1109/TMM.2021.3060951