The given dataset is a road/field dataset containing **153 images** (**108** road images and **45** field images).
The number of images is relatively small, and there is an imbalance between the number of instances of each class.
The size of images is variant, it goes from **183** pixels to **1600** pixels in height

Data augmentation:

It is a good idea to perform some data augmentation to ensure an ample number of images for training the neural network. Although augmenting data is especially beneficial in transfer learning scenarios, where a pre-trained model is fine-tuned for a new task or dataset, it's important to note that augmentations may not encompass all possible variations in real-world scenarios. Additionally, we should consider that some tasks may have specific requirements for data augmentation.

We create a new dataset by applying the following transformations:

1. Grayscaling.
2. Horizontal (180 degree) rotations.
3. Gaussian Noise (with 2 parameters)
4. Color Jitter.

The resulting dataset contains **918** images (**648** road images and **270** field images)
`*data_augmentation.py*` is the script for data augmentation, and it can be executed either by directly running the script or by calling the function `creating_file_with_augmented_images.`

Model architecture and training:

The state-of-the-art architecture, EfficientNet, has been employed to accomplish this task for the following reasons:

- It can leverage transfer learning effectively. It is pre-trained on large datasets like ImageNet and can be fine-tuned on our smaller dataset.
- It is designed to handle a range of input sizes, and its architecture incorporates a compound scaling method that allows the model to be scaled to different resolutions.
- It maintains a balance between accuracy and computational cost.

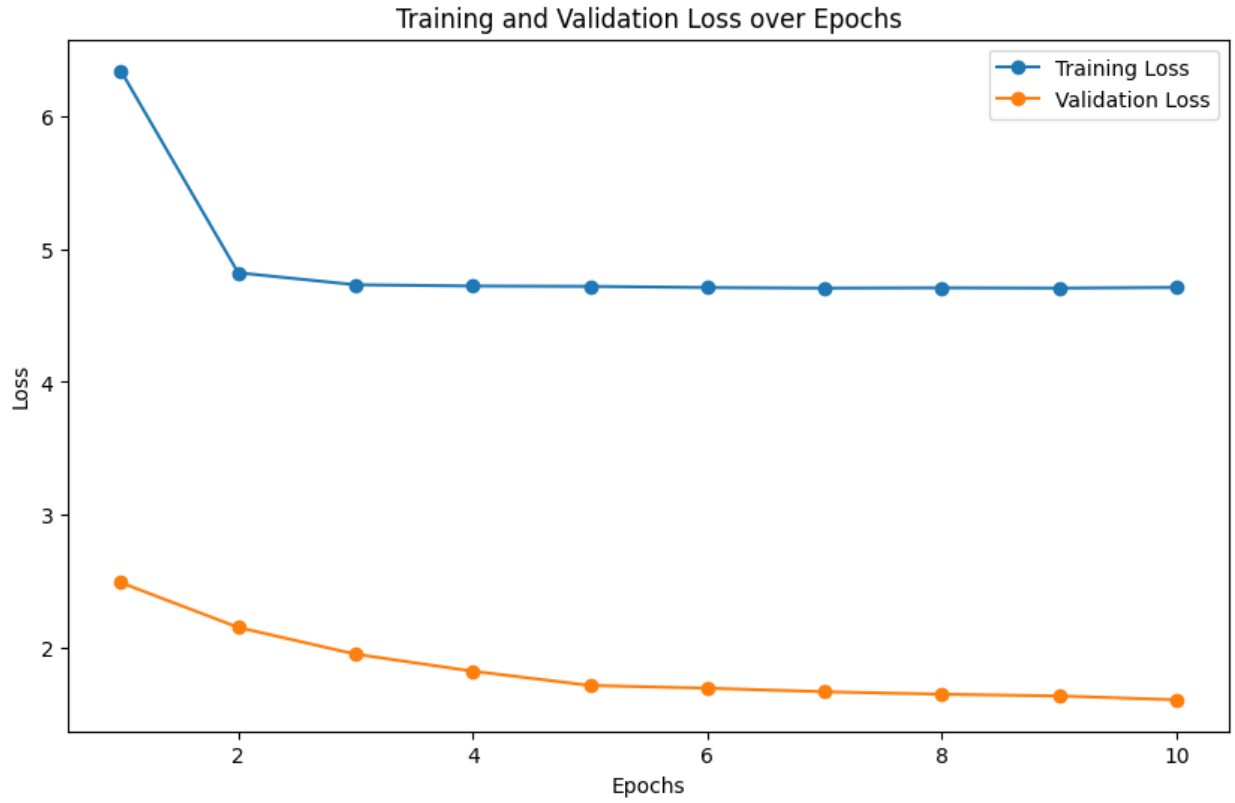
To fine-tune the EfficientNet model trained on the ImageNet dataset, we customize the classifier layers to our desired output (2 neurons with a softmax activation). We add:

1. A ReLU activation after the 1000 neurons layer.
2. A fully connected layer for the two-class output, followed by a Softmax activation.

`*model.py*` is the script for downloading and customizing the EfficientNet model.
`train.py` is the training script where everything is inside the `FieldRoadClassifier` class:

- Data Loading.
- Optimizer and loss init.
- Confusion matrix.
- Early stopping.
- Training and validation step.

Training Vs Validation Loss:



Why this configuration:

- Two neurons with softmax have been chosen for their extensibility, anticipating the potential evolution of tasks into a multi-class scenario in the future.
- Cross Entropy loss is employed for multi-class classification with a softmax activation. Class weights were introduced after observing, in the initial configurations from the confusion matrix, that one class was better predicted than the other.
- The Adagrad optimizer, featuring an adaptive learning rate, was adopted after noting in the initial configurations (using Adam) that the loss became less consistent as it approached smaller values.

Confusion matrix:

	True Positives	False Positives	True Negatives	False Negatives	Precision	Recall	F1 Score
0	109	0	40	1	1.0	0.990909	0.995434

Prediction on test images: