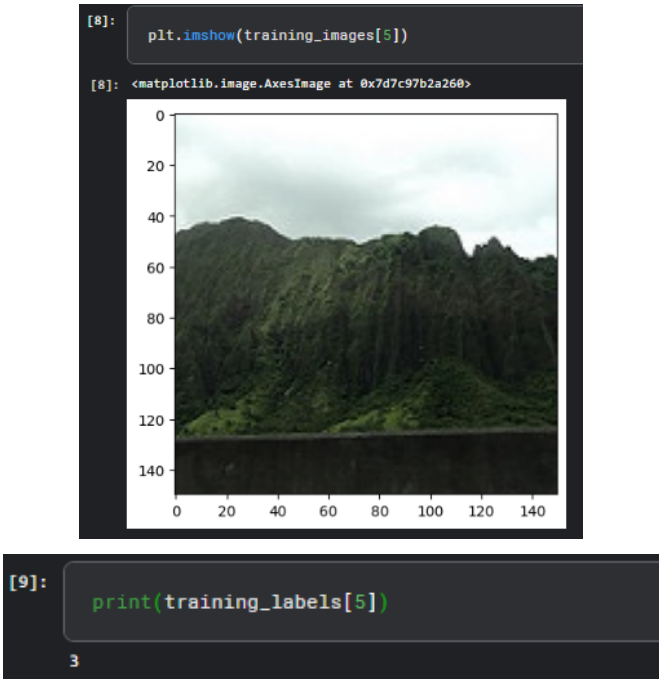
Labeling

This dataset needed some pre-processing. The images were generally labeled, since they were in categorized folders. However, for training it is necessary that each image is associated with its label, so each of the training and test images was labeled. The validation images could not be processed in this way because they were not categorized.

With this objective, the ' def labeling' was created, which also transforms the text labels to numeric labels and converts the lists in which the images and labels had been stored, into numpy arrays of type float32 and type int32. This is because working with this type of data reduces the amount of storage memory, improves model performance, and because Keras needs its input data to be of this type.

Data Visualization

In this section you can see the results of the labeling. An image of the training set is plotted and its label is printed, both are consistent.



Data preparation

This part of the code hotencodes, normalizes, and splits the data.
In the first part, it finds the number of unique classes in the training tag set and then converts the categorical tags into a one-hot encoding representation for the training and test tag sets.

In normalization: The train and test images are normalized to make sure that all images have comparable pixel values and are in a manageable range. This helps to improve the accuracy of the model and to reduce the variance of the input data.

The normalization being used here is known as "Z-score normalization" or "standard normalization". The mean and standard deviation of the training data are calculated and then used to scale both the training and test data. The formula used is:

(x - mean) / standard deviation

This normalization centers the data at zero and scales the units to have a variance of one. The constant 1e-7 is added to the denominator to avoid a possible division by zero in case the standard deviation is very small.

Finally, 10 percent of the train set is separated for validation since the set destined for validation was not labeled. We chose to do the validation in this way (training by sending the train data and validation) to save time, make better use of the data, detect overfitting problems early and optimize the overall performance of the model.

Building convnet model

Convnet architecture

Having the input images ready, these images go through a convolution network that extracts features (edges, textures, etc.) at first in a very superficial way and then, as it goes deeper into the network, much more complex features are extracted. These convolution layers are linked to a maxpooling layer that reduces complexity by limiting the length and width of the images. And so layer after layer of stacked convolutions with maxpooling will give us back an image that is smaller and smaller but deeper in its meaning and information. Next, a layer called Flatten is applied,

General parameters:

Kernel (filters)

We know that images are understood as arrays of pixels. The kernel is also a matrix (but smaller) that moves from the upper left corner to the lower right corner of the image, going step by step until it completes the entire image by doing a little mathematical operation called convolution. And in this tour, a mathematical multiplication operation is executed that obtains the data and patterns for each row and column of the image. The result of this convolution results in a new image with certain features highlighted. Thus, the objective of the filters is to detect features of the input images.

In our model we started using 32 filters in the first layer that were later increased in the following layers.

Padding
It is a margin that is added to the image so that when performing the convolution operation the resulting image does not reduce its size. 'same' is used so that it does not alter the characteristics of the original images.
In our model we use padding = 'same'.

Maxpooling
Reduces the size of the images resulting from the convolution thanks to a kernel that highlights only the most relevant features of the image.

Parameters of the optimized models:

Regularizers
We use the L2 regularization (which controls the magnitude of the weights)

kernel_regularizer=regularizers.l2(w_regularizer) is used to apply L2 regularization to the weights of a convolutional layer on a CNN. It helps to avoid overfitting and improve the generalizability of the model by penalizing large weights.

Callbacks: early stopping and checkpoints
Early Stopping:
When a neural network has stopped optimizing the accuracy or the metric that we put in 'monitor', if this metric does not rise, this can decide to end the training when the network begins to diverge.
Checkpoint:
With this we can fully execute the training network throughout the epochs that we defined but in this file the weights of the neural network that had a better accuracy are saved. This ensures that the model is always the best.

Batch normalization
It is a normalization within the hidden layers, throughout the training, since the weights vary constantly, these values can be standardized within the network. This facilitates gradient descent and works in batches. The result is standardized data even within the network training.

which brings the resulting tensor to one dimension. And having only one dimension, the classification is done by stacking dense layers as it was done in typical neural networks.

Model performances

Wrong models that generated problems in RAM

problems:

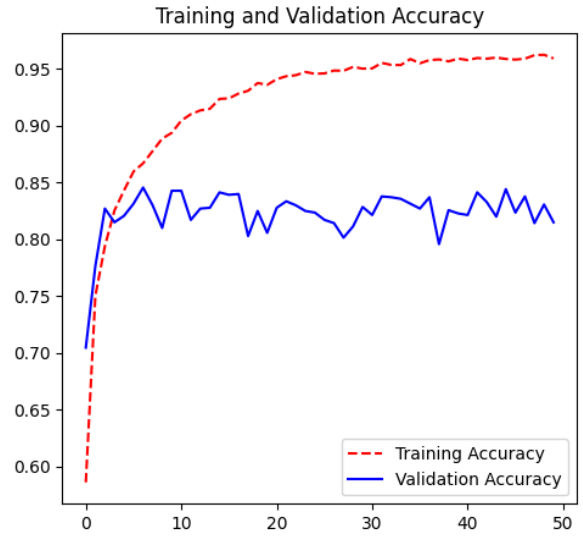
- We were not converting the image sets and labels to float32 and int32 data types

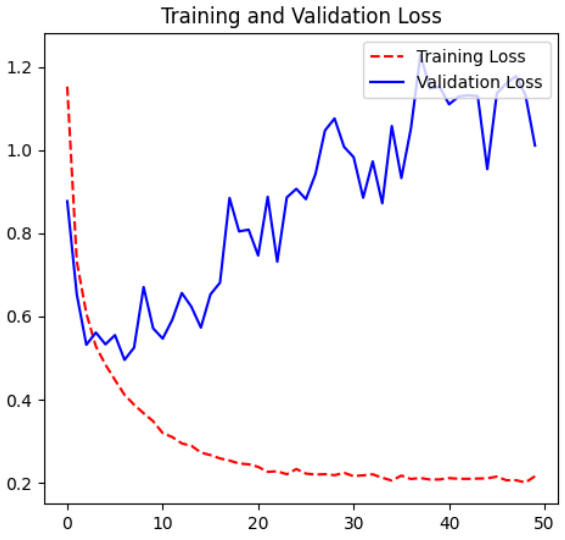
Corrected models

Model without optimizations (without data augmentation, callbacks, or batch normalization)

other features: only has a dense output layer, uses shuffle (shuffles data to introduce randomness, avoid bias and improve generalizability)

```
Epoch 50/50
395/395 - 30s - loss: 0.2156 - accuracy: 0.9591 - val_loss: 1.0109 - val_accuracy: 0.8148 -
```



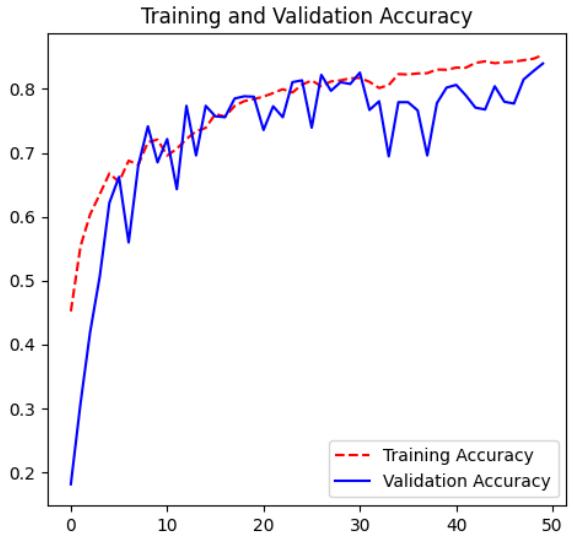


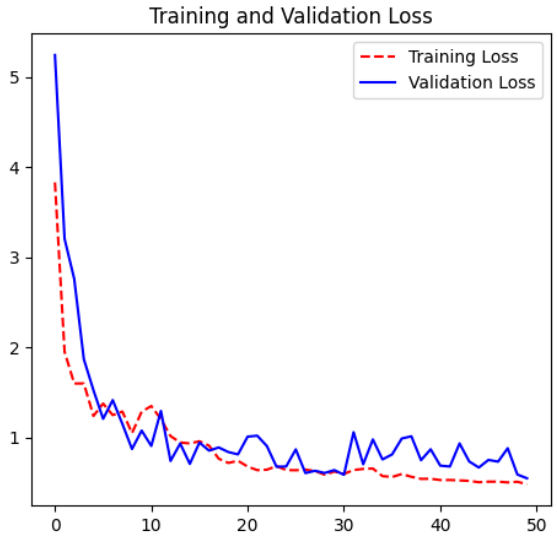

```
[28]: model2.evaluate(x_test,y_test)

      94/94 [==============================] - 2s 24ms/step - loss: 1.0902 - accuracy: 0.8097
[28]: [1.0901851654052734, 0.8096666932106018]
```

Model with optimizers

- Model with 50 epochs

```
Epoch 50/50
98/98 [==============================] - ETA: 0s - loss: 0.4797 - accuracy: 0.8530
Epoch 50: val_accuracy improved from 0.82764 to 0.83974, saving model to mi_mejor_modelo.hdf5
98/98 [==============================] - 79s 806ms/step - loss: 0.4797 - accuracy: 0.8530 - val_loss:
0.5452 - val_accuracy: 0.8397
```



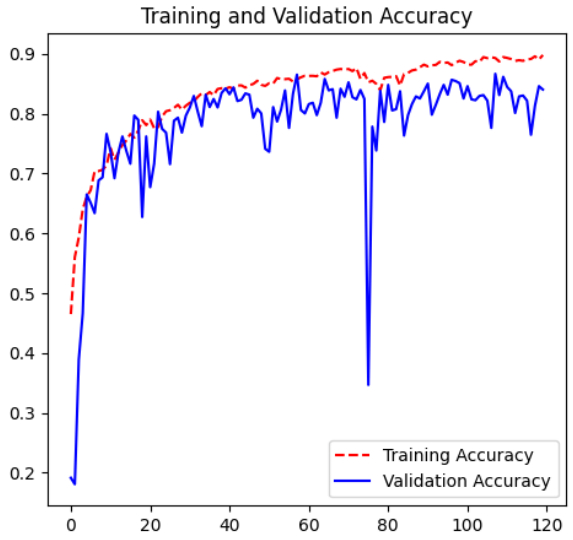


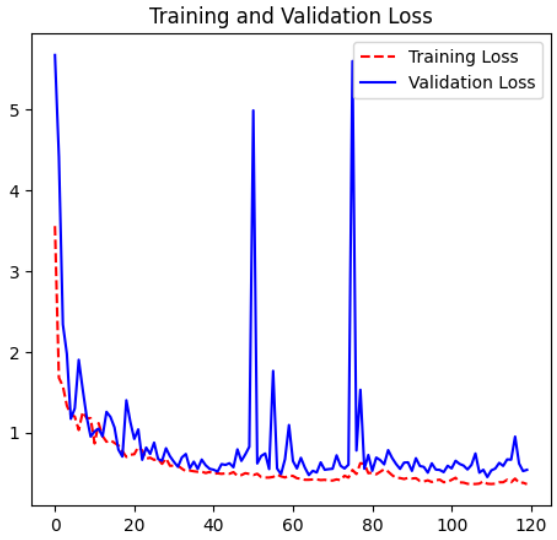

```
model2.evaluate(x_test,y_test)
```

```
94/94 [==============================] - 3s 28ms/step - loss: 0.5432 - accuracy: 0.8297
[0.5431879758834839, 0.8296666741371155]
```

- Model with 120 epochs

```
Epoch 120/120
98/98 [==============================] - ETA: 0s - loss: 0.3624 - accuracy: 0.8982
Epoch 120: val_accuracy did not improve from 0.86681
98/98 [==============================] - 81s 820ms/step - loss: 0.3624 - accuracy: 0.8982 - val_loss: 0.5403 - val_accuracy: 0.8405
```



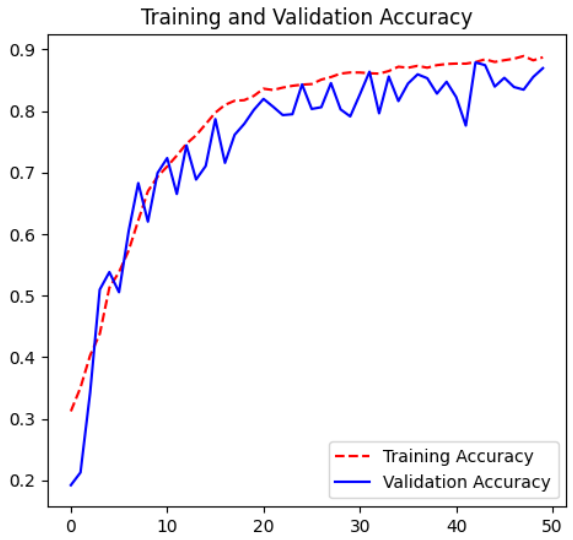


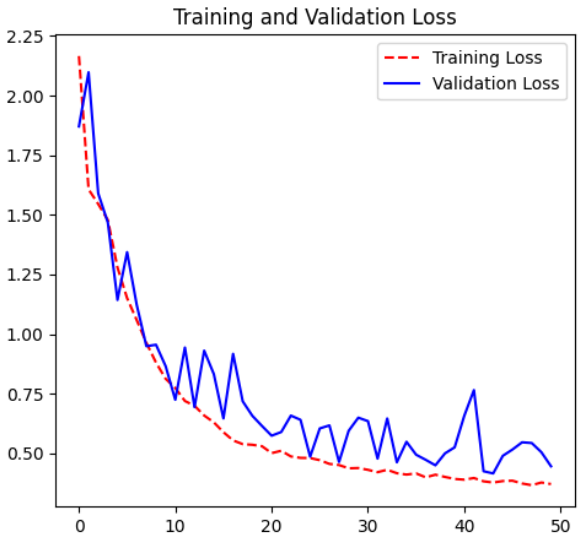
```
model2.evaluate(x_test,y_test)
```

```
94/94 [==============================] - 3s 27ms/step - loss: 0.5514 - accuracy: 0.8500
[0.5513752698898315, 0.8500000238418579]
```

- Model with 50 epochs with shuffle and with 30 dense layers

```
Epoch 50/50
98/98 [==============================] - ETA: 0s - loss: 0.3708 - accuracy: 0.8873
Epoch 50: val_accuracy did not improve from 0.87892
98/98 [==============================] - 77s 784ms/step - loss: 0.3708 - accuracy: 0.8873
- val_loss: 0.4453 - val_accuracy: 0.8697
```

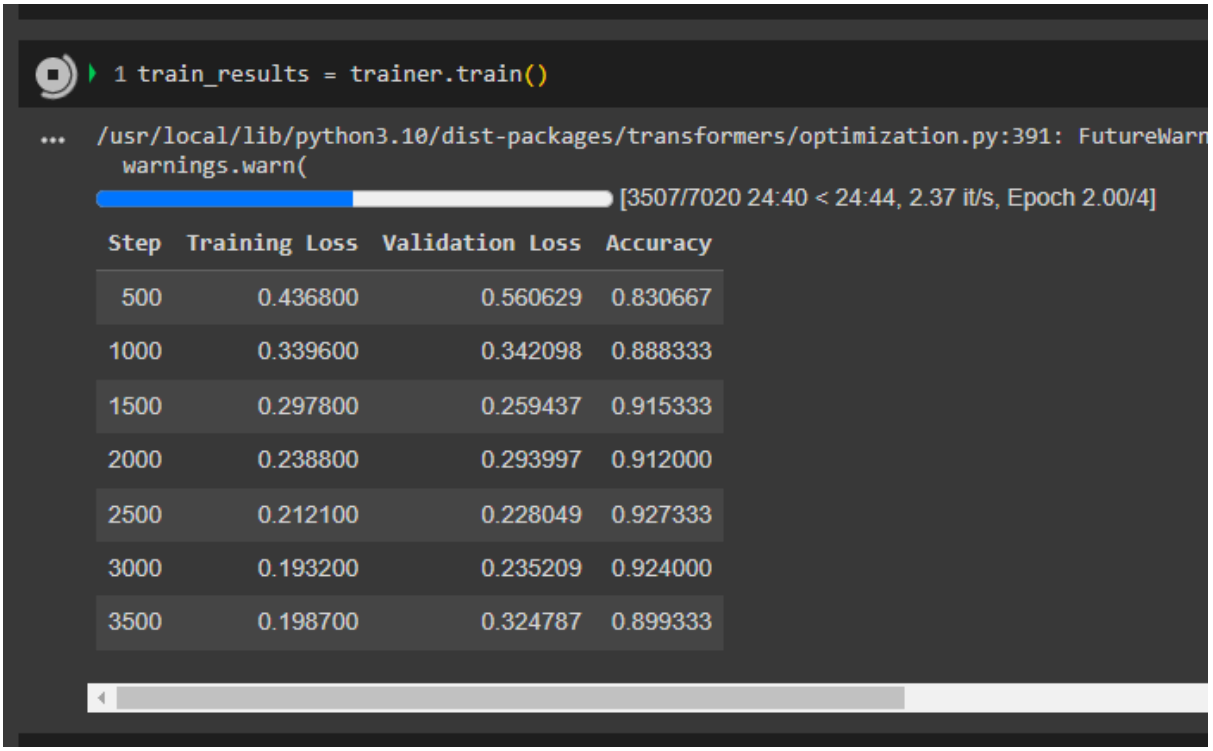





```
1 model2.evaluate(x_test,y_test)
```

```
94/94 [==============================] - 3s 28ms/step - loss: 0.5098 - accuracy: 0.8480
[0.5097863078117371, 0.8479999899864197]
```

ViT



Conclusion

Using optimizers in the model greatly improved performance, and we obtained better accuracy by using more epochs, data augmentation, and call backs. However, we were able to observe that cnn models and neural network models in general tend to overfit because in the convolution process they always obtain more specific characteristics. On the other hand, the model using Transfer Learning and ViT have the best accuracy with few times. Due to lack of time, it was not possible to carry out more tests with these models and in the case of ViT, the training was not completed, but it was observed that it had very high accuracies in the process, just like the model with transfer Learning.