Image Recognition for Prediction of Forest Health

Moustafa Elbery
Matr.: 259703

Sebastian Hösing
Matr.: 259734

Ding Ken Chuah
Matr.: 256558

***Abstract*—[M.Elbery] The project aims to efficiently analyze forest health and damage using machine learning. The goal is to classify the health status of individual trees into four classes: High Damage, Low Damage, Healthy, and other. A dataset is provided that contains satellite images from two different sets of forests. Different ways of image pre-processing were investigated in order to prepare the data for ML algorithms. The methods used for classification are Convolutional Neural Network, k-Nearest Neighbor classifier, and Support Vector Machine. A comparison between the different models was investigated according to the performance and accuracy of results. It was apparent that the CNN was the best classifier for this problem.**

I. Introduction [D.Chuah, S.Hösing]

The classification of the health status of trees has become very important due to problems such as climate change and pollution. Large amounts of data are available in the form of satellite images. However, it is infeasible to classify all trees manually. If this data can be classified using machine learning, it could be used for continuous monitoring or further studies on the effects of environmental crises. In this project, the applicability of Convolutional Neural Networks, the k-Nearest Neighbour classifier, and Support Vector Machines to this problem was investigated.

An overview of the structure of the project is shown using the flowchart in Figure 1.

II. Methodology

A. Image Pre-processing [D.Chuah, M.Elbery]

Annotated image data from two different sets of forests were prepared and processed to be usable for training of a classification model. The steps involved loading, cropping, resizing, and extracting of the labels. The test data were processed the same way but was only used for evaluating the models at the end. All data labels were converted to one-hot-encoding format. The training data were then split into 80% for training and 20% for validation [1], in addition, a copy of the data was converted to grey-scale.

The following pre-processing parameters of the images were investigated: image size, interpolation method for resizing, omitting truncated images, sharpening for small images. For the training and test forest images, the respective coordinates of the tree images are cropped based on the data from the annotation. To ensure equal dimension, all the extracted tree images are resized into 100 x 100 pixels (ref. Fig. 2).

After extracting the single tree images from a forest image, some of the tree images have very small dimension, which leads to blurry images after resizing it. An image sharpening filter kernel is used to filter the blurry image if the original image is less than a certain threshold of dimension. (ref. Fig. 3)

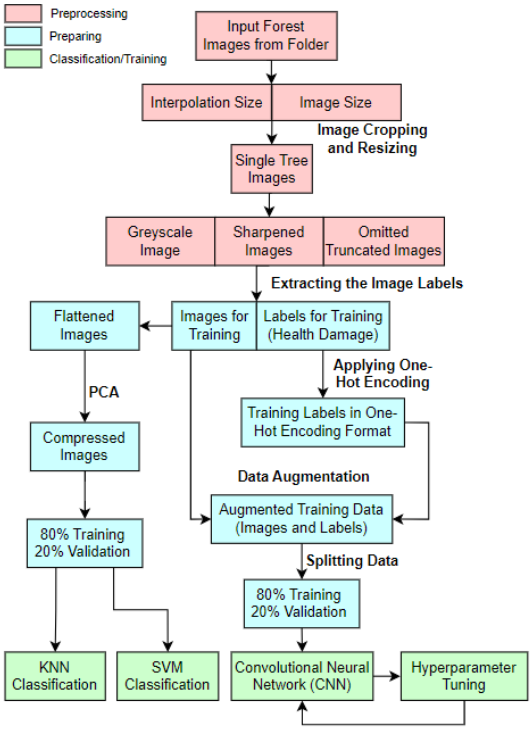


Fig. 1: Flowchart of Image Recognition

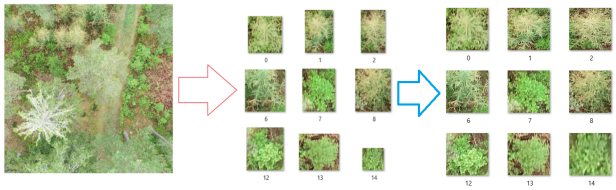


Fig. 2: Resultant Images after Cropping(Middle) and Resizing(Right)

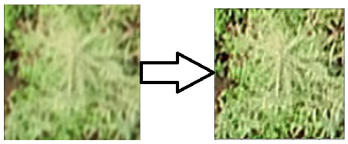


Fig. 3: Original(Left) vs Sharpened Image(Right)

B. Preparing the data

1) Principal Component Analysis (PCA) [S.Hösing]: The toolbox "sklearn.decomposition.PCA" is used for the dimensionality reduction [2]. The images are converted to greyscale and flattened into arrays. The dimensionality reduction is

applied to this vector. Different numbers of principal components, ranging from 1 to 4501 in 500 steps, are applied to check the explained variance ratio. This is necessary because the SVM and the KNN only take a 1-D vector for each sample as input. Using the complete vector as input for the SVM is computationally too expensive. The KNN algorithm requires tuples for each samples as input.

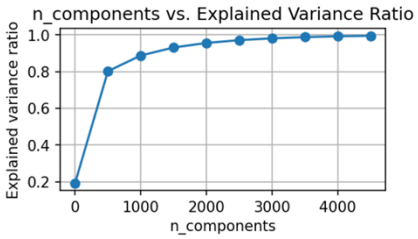


Fig. 4: Number of principle components vs. the explained variance ratio for the test data set

2) Data Augmentation [M.Elbery]: After analysing the training data, it was clear that the data was unbalanced:

Class (Damage Type)	HD	LD	H	other
# of training images	3,215	12,707	1,659	2,941

In order to balance the data and prevent bias during training, data augmentation was used to generate new images from the original ones. To achieve this, the *ImageDataGenerator* class in the '*tensorflow.keras.preprocessing.image*' library was used [3]. The new generated images are randomly rotated and flipped (horizontally and vertically) images of the original training images. To avoid aggressive augmentation that may lead to loss of features, minimal augmentation parameters were used. Since '*LD*' class already has 12,707 images, this was only done to the images corresponding to other classes. The result is a balanced training dataset with 50,828 (12,707 x 4) images of individual trees with their corresponding labels.

An alternative way to counteract the imbalanced data is using weighted sampling during training. The weights for each class are chosen heuristically. Samples of HD are weighted with 1.8, of LD with 1, of H with 10 and of other with 3. An array with the corresponding weight of each sample is passed to the CNN training [S.Hösing].

C. Classification Models Implementation

1) Support Vector Machine (SVM) [S.Hösing]: SVM is a supervised algorithm used that can be used for classification and regression. It finds a hyperplane in an N-dimensional space that separates the data points of different classes. The margin between the closest data points to the hyperplane is maximized. [4]

The toolbox *sklearn.svm.SVC* is used for implementing the SVM. The SVM cannot directly classify multiple labels. Two options for decision functions, *one vs. one* and *one vs. rest*, are explored. If the parameter *class_weight* is set to *balanced*, a regularization is applied on the loss function for each class. The weight for the regularization is based on the class frequency [5]. 500 principal components are used. 80.2% of the information (ref. Fig. 4) is retained at a compression of 95%.

2) K-Nearest Neighbour (KNN) [D.Chuah]: K-Nearest Neighbour works such that it computes the Euclidean distance between the feature vectors, which are the images themselves, and classifies the images based on the K number of neighbours between the feature vectors. Initially, the images are reduced to a dimension of 2 using PCA. 24.5 % of the information is retained at a compression of 99.98%.

3) Convolutional Neural Network (CNN) [M.Elbery]: CNNs have revolutionized the field of image classification and computer vision due to their ability to automatically learn spatial hierarchies of features from input images. Some of the key advantages are: automatic feature extraction, parameter sharing, and adaptability to different image types. These advantages make CNNs a powerful tool for image classification, enabling high accuracy and robust performance in categorizing images [6].

The CNN was created using *keras* in *tensorflow* library. The functional API was used to implement the CNN [7]. The optimizer used was Adam with a learning rate of 0.0003, and the loss function is Categorical Cross-entropy, which is widely used for classification tasks. Figure 5 is a block diagram of the network architecture:

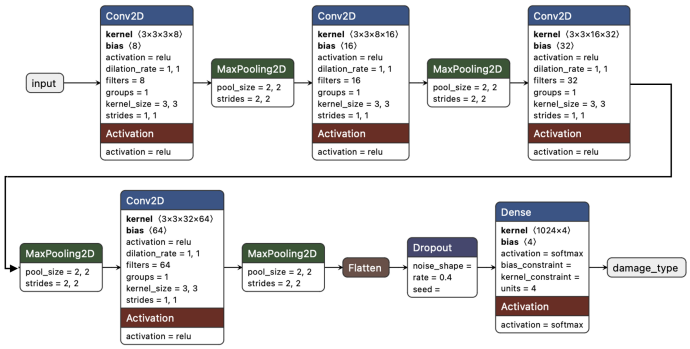


Fig. 5: CNN Architecture

During training, a Custom Model Checkpoint [8] was used to save the model with best validation accuracy. Shuffling of the training data was also implemented at the beginning of each training epoch, which ensures randomized batches during training, leading to better model generalization and performance. A batch size of 32 was chosen and the CNN was trained for 50 epochs.

Hyper-parameter tuning of the following was done:

- Structural hyper-parameters: number of layers (Convolutional, Max-Pool, Dense) with their respective neurons, and dropout rate.
- Training hyper-parameters: learning rate, batch size, and number of epochs.
- Pre-processing Parameters: image size, omitting truncated images, interpolation method for resizing, and image sharpening for small images.

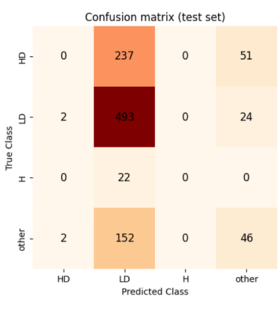
III. Results

The main metric used for validating the different classification models was the validation accuracy, as well as visual analysis of a confusion matrix.

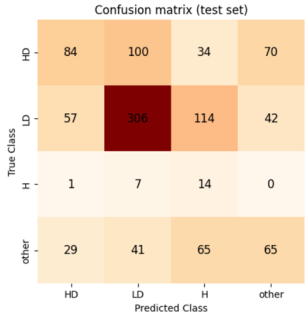
A. Support Vector Machine (SVM) [S.Hösing]

For both decision functions, the validation accuracy for the fitted SVM without regularization is 53.8% percent. Evaluating the SVM on unseen data yields an accuracy of 52.7%. Almost all test samples are predicted as *'LD'* or *'other'* (ref. Fig. 6).

Using regularization, a validation accuracy for both decision functions of 53.6% is achieved. Evaluating the test dataset yields an accuracy of 44.5% for the one vs. one approach. The prediction accuracy for the *'HD'*, *'H'* and *'other'* class has slightly improved, while the prediction accuracy for the *'LD'* class has decreased (ref. Fig. 6).



Fitted without regularization (Accuracy: 52.7%)



Fitted with regularization (Accuracy: 44.5%)

Fig. 6: Confusion Matrix of the fitted SVM for unseen data

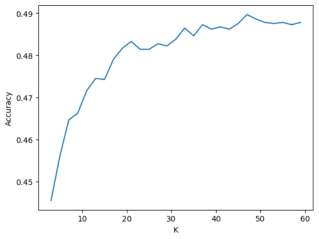
Without regularization the SVM is not fitted to the features. It instead only predicts *'LD'* and *'other'*, because they are more common than the other classes. Using regularization better generalization is achieved, but there is the tendency to misclassify *'LD'* trees as *'H'*. This makes sense, since the features of a slightly damaged and a healthy tree should be similar. The overall poor performance is probably a result of the loss of color information and spatial relations in the preprocessing.

B. K-Nearest Neighbour (k-NN) [D. Chuah]

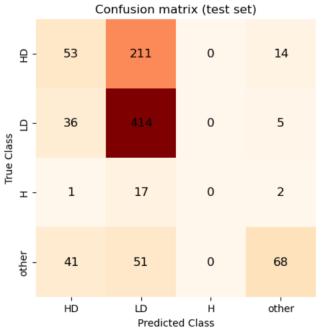
Since there are training and test image datasets, 80% of the original training datasets are split into training datasets and 20% of them goes to validation datasets. Splitting the datasets helps preventing overfitting, such that the model performs well in training data and unseen data. To determine the value of K, the accuracy vs the K value is plotted (ref. Fig. 7).

The accuracy reaches a peak value between K=40 and K=50. However, since the accuracy does not improve significantly after K=40, this value is chosen as the optimal value. Evaluating the K Nearest Neighbour on the test dataset yields an accuracy of 53%.

Looking at the confusion matrix (ref. Fig. 7), it is observed that most of the test samples are predicted as High Damage (HD) or Low Damage (LD), with very little of them predicted as Other, and none of the test samples are predicted as Healthy (H). Additionally, K-Nearest Neighbours show that the prediction works decent on Low Damage classification, but it struggles to predict the High Damage accurately, and it cannot predict any H classification at all.



Graph of Accuracy vs K value for health classification



Confusion Matrix of the k-NN classifier for unseen data

Fig. 7: k-NN results

C. CNN Results Analysis [M.Elbery]

1) Hyper-parameter Tuning: The main metric used to compare between models with different hyper-parameters is the validation accuracy. Tuning was done using grid search for the most promising regions.

- Structural hyper-parameters:
  - Number of layers and neurons:

# Layers and Neurons	3 Conv (32,64,128) + 3 Max-Pool + 2 Dense (64,4)	3 Conv (16, 32, 64) + 3 Max-Pool + 1 Dense (4)	4 Conv (8, 16, 32, 64) + 4 Max-Pool + 1 Dense (4)
# Trainable Parameters	433,508	49,188	28,628
Training time (30 epochs)	$>$ 4 hrs	2 hrs	40 min
Validation Accuracy(%)	76.6	75.82	82.93

    As one can deduct from the table, having a more complex architecture with a high number of convolutional channels (filters) and dense neurons will not improve the performance but will increase the training time and model complexity.
  - Dropout rate: Having a dropout rate of 0.4 for the dropout layer showed the best accuracy results and to prevent overfitting.
- Training hyper-parameters:
  - Learning Rate:

Dropout rate	0.0001	0.0003	0.0005	0.0008	0.001	0.0015
Validation Accuracy	81.78	82.93	82.32	81.55	80.50	78.30

    Having a learning rate less than 0.0001 will result in very slow training and little improvement, whereas, a learning rate bigger than 0.001 will negatively affect convergence.
  - Batch Size: In general, having a lower batch size increases generalization capabilities, however, it increases training time. Choosing a batch size of 32 shows a good balance between good generalization and moderate training time.

- Number of Epochs: Less epochs for training is not enough to achieve a good validation accuracy. On the other hand, training it for a lot of epochs can risk overfitting. A balance between number of epochs, learning rate, and dropout rate should be achieved. Training the CNN for 50 epochs showed the best results for convergence and generalization.

- Pre-processing Parameters:
  - Image size: A larger image size will drastically increase the complexity of training, whereas, a smaller image size can risk loss of features in the input images. Choosing an image size of 100 by 100 proved to provide a good balance.
  - Omitting truncated images: Tree images that are on the frame edges of the original forest images might be truncated. Choosing to remove them from the training data did not show significant improvement.
  - Interpolation type for resizing: Choosing a cubic interpolation method for resizing showed the best performance.

2) CNN training: This section shows the training results for the CNN, trained using balanced and augmented data. However, training the CNN with the original training data shows bad generalization for some classes since the data is unbalanced.

After tuning the hyper-parameters, a decision to do data augmentation for the training data was made to increase balance and coherence of the data. The CNN was trained using the following optimal hyper-parameters:

Learning Rate	Dropout Rate	Batch size	Epochs	Image size	Omit truncated	Interpolation type	Image sharpening
0.0003	0.4	32	50	100x100	False	Cubic	True

Optimal Hyper-parameters used for training CNN

Figure 8 shows the training/validation loss and accuracy for 50 epochs using balanced and augmented data:

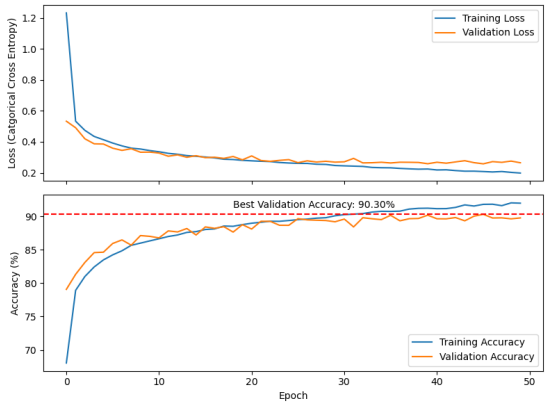
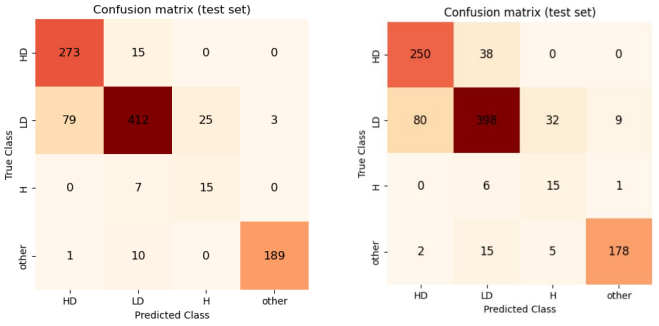


Fig. 8: CNN training/validation loss and accuracy for 50 epochs with balanced training data using Data Augmentation

The best validation accuracy reached during training for the CNN using the previously mentioned hyper-parameters with balanced data using augmentation is 90.3 %. Another way to balance the data is using weighted samples, training the CNN using weighted sampling results in an accuracy of 81%. Weighted sampling did not improve the test accuracy. Perhaps more aggressive weights for the underrepresented classes are required. The main advantage of data augmentation over weighted sampling is that it actually balances the training data so that each class has an equal number of input images. This results in a much higher validation accuracy, also because of the extra features learned by the model for each augmented image. However, generalization capabilities of the CNN using both methods is improved and it especially shows for the *'H'* class since it had the least amount of training data originally.

Figure 9 shows CNN test confusion matrix with the test (unseen) data for both methods used to balance the data:



Trained with balanced data using Data Augmentation (Accuracy: 90.3%)

Trained with weighted samples (Accuracy: 81%)

Fig. 9: Confusion Matrix of the trained CNN for unseen data

D. Comparative Study [M.Elbery]

Table I shows a comparison between the diffrent classification models.

Model	k-NN	SVM	CNN
Validation Accuracy (%)	53	44.5	90.3
Generalization Capability	Bad	Moderate	Very Good
Model Complexity	Low	Low	High

TABLE I: Comparison between different classification models

IV. Conclusion [M.Elbery]

In conclusion, employing machine learning models, particularly classification techniques, to analyse the health status of forests is a very efficient approach to do so. Processing and preparing satellite image data to make it suitable for model training is a crucial step. Ensuring that training data is well-balanced, extensive, diverse, and representative of your actual environment is very important in order to ensure that your training is effective [9] and that your model can generalize well to unseen data with high accuracy. After comparing various models, it is evident that CNNs are the most reliable at tackling this problem. Hyper-parameter tuning, while challenging, can significantly enhance performance when executed correctly.

REFERENCES

[1] **The Importance of Data Splitting** *https://mlu-explain.github.io/train-test-validation/*, visited on 17.07.2024

[2] **scikit-learn documentation** *https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html*, visited on 15.07.2024

[3] **Image Data Generator - tensorflow documentation** *https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator*, visited on 03.07.2024

[4] **Support Vector Machine (SVM) Algorithm** *https://www.geeksforgeeks.org/support-vector-machine-algorithm/*, visited on 17.07.2024

[5] **scikit-learn documentation** *https://scikit-learn.org/stable/modules/generated/sklearn.decomposition. PCA.html*, visited on 15.07.2024

[6] **Convolutional Neural Networks (CNNs) in Computer Vision** *https://medium.com/@AIandInsights/convolutional-neural-networks-cnns-in-computer-vision-10573d0f5b00*, visited on 17.07.2024

[7] **CNN Functional API - tensorflow documentation** *https://www.tensorflow.org/guide/keras/ functional_api*, visited on 10.06.2024

[8] **Model Checkpoint - keras documentation** *https://keras.io/api/callbacks/ model_checkpoint/*, visited on 15.06.2024

[9] *Dablain, D., Jacobson, K.N., Bellinger, C. et al. Understanding CNN fragility when learning with imbalanced data. Mach Learn 113, 4785–4810 (2024).https://doi.org/10.1007/s10994-023-06326-9*