Universiteit Leiden
The Netherlands

Bio-informatics

The performance of RetinaNet and Faster R-CNN with classification on 3D pollen images

Tijs Konijn

Supervisors:
Lu Cao & Fons J. Verbeek

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

01/08/2024

Abstract

The accurate classification of pollen grains is critical in various fields such as palynology due to its implications on allergy forecasts and ecological studies. This research focuses on the classification of the Urticaceae family, particularly the genera Urtica and Parietaria, which are morphologically similar yet differ significantly in allergenic potential. Traditional methods of pollen classification are time-consuming and require expert knowledge. This study explores the application of using 3D images for deep learning techniques to automate the classification process, potentially improving and enhancing upon 2D image classification in both speed and accuracy.

This research utilized a dataset comprising 6472 pollen stack images, each with 20 slices along the Z-axis, to train several deep learning models, including ResNet3D, Faster R-CNN, and RetinaNet. The models were optimized through transfer learning and various frame selection strategies. The results demonstrate that the deep learning models significantly outperform traditional machine learning methods, achieving high accuracy rates. The pre-trained ResNet3D model, using optimal frame selection and extended epochs, achieved best with an F1-score of 98.3% and an accuracy of 98.3%.

The findings indicate that increasing the number of frames beyond a certain threshold does not significantly enhance performance, suggesting an optimal frame selection strategy is more effective. Additionally, while extending the number of training epochs marginally improves accuracy, the trade-off in computational resources must be considered.

Contents

1 Introduction

Pollen seasons are extended because of global warming and climate change [DCNMO+20]. This is leading to an increase of hay fever patients that are affected by the rise in allergenic pollen levels in the air [BWT+19]. This is the reason that classification of pollen is important for many fields, such as palynology that need this information to provide safety treatments to patients affected by allergenic rhinitis. There are a lot of different pollen families, which all have very different levels of inducing allergies. Thus it is of great importance to be able to correctly classify these pollen by the difference in characteristics between pollen grains.

This paper will be focusing on a specific example, the nettle family Urticaceae. This family consists of two genera, Urtica and Parietaria, that on a morphological level are very similar. These pollen are very common in Europe. The only difference between the two genera is that Parietaria is highly allergenic and Urtica is not close to being nearly as harmful. This means that a forecast of all the levels of pollen grains in the air can be very important, especially when global warming and climate change result in extended pollen seasons [DCNMO+20].

Currently airborne pollen are counted through the use of a microscope. Finding the difference between different pollen grains is however challenging with the pollen grains being visually very similar. A possible solution to this problem is automatic classification through the use of deep learning. These deep learning-based techniques not only have the potential to be faster at classifying pollen grains compared to a palynologist, but it might also do this with better accuracy. Automated pollen analysis has already been shown to offer great benefits. It was shown that with simple image processing techniques the pollen counting speed was significantly increased. The method that was used took about 60 seconds per image, while a human would need between 5 and 68 minutes to count [CY09].

1.1 Literature Review

Machine learning methods for pollen classification traditionally require manually selected features to extract them from images. These features are based on shapes, texture and other properties that are related to pollen grain images [LPC+23]. In contrast, deep learning methods automate this feature selection through convolutional layers. Many state-of-the-art Convolutional Neural Networks (CNNs) have been applied to pollen classification tasks before. For instance, a study by Rostami et al. [RBD+23] applied various pre-trained CNN models, including AlexNet, VGG-16, MobileNet-V2, ResNet (18, 34, 50, 101), ResNeSt (50, 101), SE-ResNeXt, and Vision Transformer (ViT), to classify pollen grains from the Great Basin Desert. The dataset consisted of 10,000 images of 40 pollen species, and the ResNeSt-110 model achieved the highest accuracy of 97.24%. Another study by Daood et al. [DRB16] used a seven layer CNN model to classify 30 pollen types, achieving a classification rate of approximately 94%. A study by Astolfi et al. [AGM+20] compared eight different CNN models, which are DenseNet-201, Inception-ResNet-V2, Xception, Inception-V3, VGG16, VGG19, ResNet50 and NasNet, on 73 pollen categories. They found that ResNet50 with an accuracy of 94.0% and DenseNet-201 with an accuracy of 95.7% performed best.

In a paper written by Chen Li et al., the accuracy of machine learning and deep-learning based methods were discussed [LPC+23]. Different methods for classifying the Parietaria and Urtica pollen grains were tested on accuracy. It was found that deep-learning based methods performed better compared to machine learning methods, with the best model achieving an accuracy of 99.4% compared to the best machine learning model with an accuracy of 94.5%. The neural network architectures that were used for this comparison are AlexNet, VGG16, VGG19, MobileNet V1, MobileNet V2 and ResNet50. ResNet50 is the one that performed best and therefore this will also be the model that will be used for the backbone of RetinaNet and Faster R-CNN.

The dataset that was used for this paper consists of 6472 pollen stack images [LPC+23]. Every pollen grain image in this dataset was captured with 20 slices along the Z-axis. This stack of grey-scale images for one pollen grain is because the pollen grains are three dimensional, which means that it is hard to set a focal plane for the pollen samples. Figure 1 depicts how the images were processed. The images are processed into three different projections because not all 20 slices in the Z-stack are in in-focus, therefore these images are processed so that the most informative features possible can be obtained. The projections that were used are Standard Deviation (STD), Minimum Intensity (MIN), and Extend Focus (EXT). These images are also padded because same-sized images are required for the classification models to work.

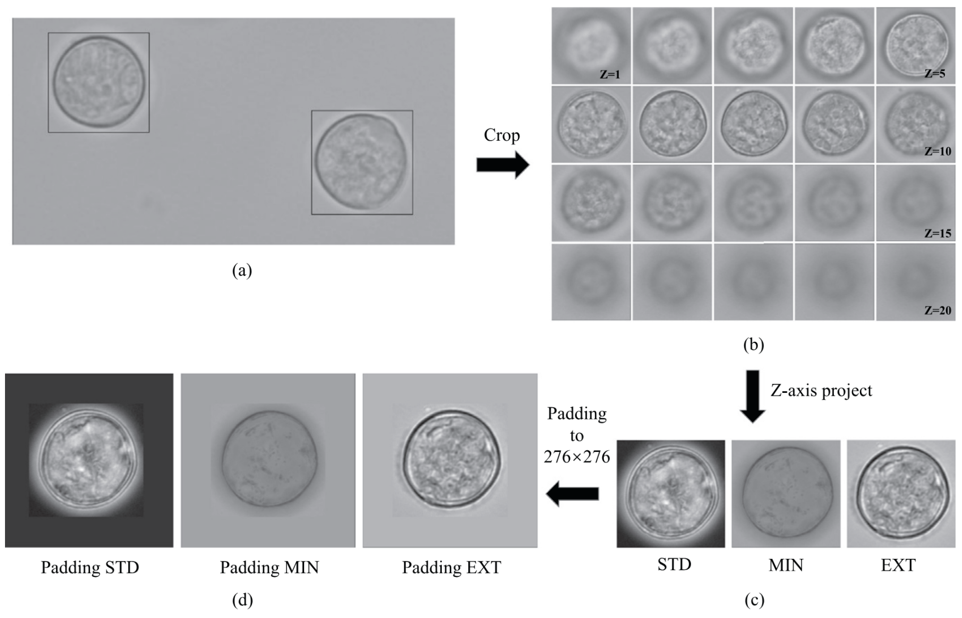

Figure 1: Workflow of pollen image acquisition of Paper Chen Li et al.. (a) One slice of the raw pollen image. (b) Gray scale image of all 20 planes in one pollen grain image (c) The different projections where STD = Standard Deviation Projection, MIN = Minimum Intensity Projection and EXT = Extend Focus Projection (d) The projections with a padded image [LPC+23].

The paper of Shreya investigates if a possible accuracy gain can be found by using a 3D convolutional model . In this paper all 20 planes were used instead of the three projections used by Chen Li et al.. This was done through a ResNet3D model, which utilizes a 3D convolutional network. It was found that this model performed best on the whole stack, achieving a classification accuracy of 95%

[Seb23]. This paper however tried to distinguish if smaller central subsections of the stack would perform better. This central plane is however not always the plane in focus. Therefore in this paper it will also be tested if an in-focus focal plane as the central subsection will make a difference to this conclusion.

Gallardo et al., has created a method for multi focus pollen detection [GGOGV+24]. They were able to create a system that is able to classify and locate a pollen grain in one step. It does so by processing the pollen grain samples in the z-axis which allows it to increase the probability of detecting the pollen grain compared to one image per sample. They have tested the performance of two deep learning techniques, RetinaNet and Faster R-CNN. The system was trained to recognize 11 pollen types that were taking through light microscopy. They were able to locate the pollen with 97.6% success and these pollen were able to be classified with an accuracy of 96.3%. This paper was able to gain better accuracy on 3D images compared to the ResNet3D model, therefore these models will be compared to each other in classification accuracy when tested on the same dataset.

1.2 Thesis overview

The approaches discussed in Section 1.1 have not extensively explored the use of 3D images, particularly those derived from the central subset of the Z-stack, for pollen classification. It also opens up questions if the different neural networks mentioned could find a possible improvement on grey scale images. This is what will be discussed in this report. It will also acquire a more robust accuracy score through the use of K-fold cross-validation. The dataset will first be preprocessed by finding the optimal frame section through edge detection after which a few different models will be tested on accuracy. The time that a model takes to train will also be investigated since the models has no need to be very big. The three models that will be used are ResNet3D as a baseline and RetinaNet and Faster R-CNN as possible improvements to this neural network. The RetinaNet and Faster R-CNN are object detection and classification models. This means that these models are also able to detect an object, which is not necessary for this dataset. Therefore the only parts of these networks that will be used are the image classification parts. These models will use two different backbones, a ResNet50 backbone and the ResNet3D network as backbone.

2 Definitions

Convolutional Neural Networks (CNN's) are a class of deep learning models that are primarily used for analyzing visual data. They are particularly effective for tasks such as image classification, object detection, and facial recognition [IBM].

The inspiration for CNN's comes from the biological visual system, specifically the human visual cortex. In the visual cortex, neurons respond to stimuli in a restricted region of the visual field known as the receptive field. Similarly, in a CNN, each neuron receives input from a small region of the input data. This local connectivity pattern allows the network to focus on local features in the early layers, while later layers can combine these local features to detect larger, more complex patterns [Kei23].

Another key aspect of the visual cortex that has inspired the design of CNN's is the hierarchical organization of neurons. Neurons in the early stages of the visual pathway respond to simple features like edges and bars, while neurons in later stages respond to more complex patterns like shapes and objects. This hierarchical organization is mirrored in the architecture of CNN's, where early layers detect simple features and later layers combine these features to detect more complex patterns.

2.1 Convolutional Neural Networks

A CNN has a few key components [Kei23]. These components are:

1. Convolutional Layer: This is the first layer of a CNN. It applies a set of filters to the input image to create a feature map. The filters are small matrices that move across the image, performing a dot product with the part of the image they are currently on, and then storing the result in the feature map.

2. ReLU (Rectified Linear Unit) Layer: This layer applies the ReLU activation function to the feature map, introducing non-linearity into the model. The ReLU function is defined as

$$f(x) = \max(0, x) \tag{1}$$

, which means it sets all negative values in the feature map to zero.

3. Pooling Layer: This layer reduces the spatial size (width and height) of the feature map, which helps to decrease the computational complexity of the network. It does this by applying a pooling operation (such as max pooling or average pooling) to patches of the feature map.

4. Fully Connected Layer: This is the final layer of a CNN. It takes the output of the previous layers, flattens it into a single vector, and then performs a simple linear operation on it. The output of this layer is then passed through a softmax function to produce the final classification probabilities.

A loss function and optimizer can be used in the process of training a CNN. A loss function in a neural network is a mathematical tool used to quantify the difference between the predicted output of the network and the actual target values. It serves as a crucial feedback mechanism, guiding the network's learning process during training. By calculating the error between the predicted and true outputs, the loss function produces a scalar value, which the optimization algorithm (typically stochastic gradient descent) then uses to adjust the network's weights and biases. The goal is to minimize this loss over time, leading the network to make increasingly accurate predictions. Different types of loss functions are used depending on the task, such as mean squared error for regression or cross-entropy for classification [Opp22]. The choice of loss function directly impacts the network's performance and the efficiency of the learning process.

Optimizers are algorithms that play a pivotal role in the training of deep learning models. Their primary function is to adjust the parameters of the model, such as weights and biases, to minimize the loss function, thereby improving the model's performance on given tasks. One of the most fundamental optimizers is Stochastic Gradient Descent (SGD), which updates parameters in the opposite direction of the gradient of the loss function with respect to the parameters. Another widely used optimizer is Adam, which stands for Adaptive Moment Estimation. It combines the advantages of two other extensions of SGD, AdaGrad and RMSProp, to handle sparse gradients on noisy problems. RMSProp, or Root Mean Square Propagation, is an optimizer that utilizes the magnitude of recent gradients to normalize the gradients. This is particularly useful in RNNs as it combats the vanishing or exploding gradient problems by adapting the learning rate [Gup24]. The choice of optimizer can significantly affect the speed and quality of the training process.

2.2 3D CNN

A 3D Convolutional Neural Network is a type of CNN that is designed to process 3D data. Instead of 2D convolutions, it uses 3D convolutions, allowing it to capture spatial and temporal information from the data. This makes 3D CNN's particularly useful for tasks such as video analysis and medical imaging, where the third dimension (time or depth) carries important information. The key difference between a 2D CNN and a 3D CNN is that a 3D CNN takes in a 3D volume as input (e.g., a video or a 3D medical image), and its filters are also 3D. This allows the network to learn features across the depth of the input volume, in addition to its width and height [Ten].

3 Methodology

The project uses a wide range of python libraries, a few of the most important ones are cv2 (OpenCV), pandas, numpy and Pytorch. These datasets are used for things such as the creation and training of the model and for preprocessing data. There are a lot of hyperparameters that can be changed for testing. This project will focus on the settings for optimal amount of frames, amount of epochs and pre-trained and non pre-trained 3D CNN models.

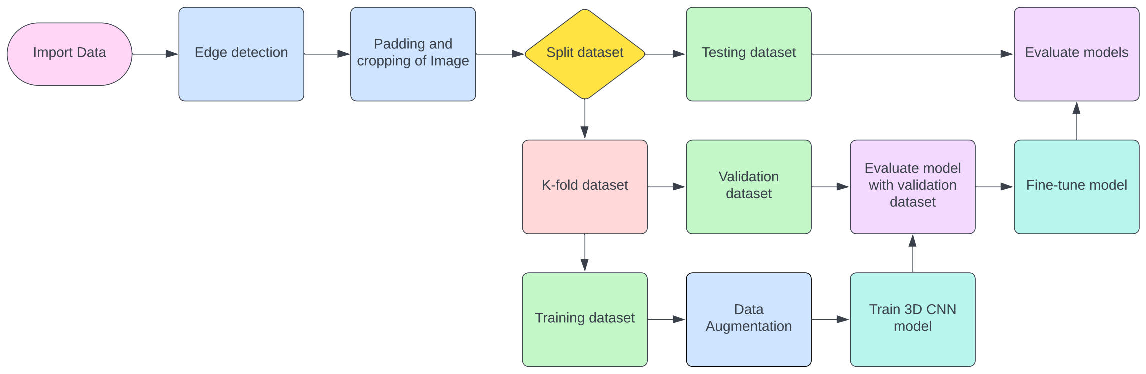

Figure 2: Flowchart of testing methodology

First the pollen are loaded by obtaining all their file paths. The dataset is sorted by folders. In total there are 5 folders, each representing a pollen species. These folders are called *P_judaica*, *P_officinalis*, *U_dioica*, *U_membranacea* and *U_urens*. These represent their respective species which is furthur explained in section 3.1. They are also merged the same way as explained in this section, which was because of their allergy level. These three class labels are then converted to an integer, after which the dataset is shuffled.

The data preprocessing has a few steps. First the edge detection is performed to find the optimal frame. Then these frames are padded and cropped to 224x224 pixels. This threshold has been chosen because the majority of the dimensions from the images in the dataset fall below it. The subsection of frames that is chosen depends on the amount of frames allowed. This difference in performance will be tested for 4, 6, 8 and 20 total amount of the optimal frames. Then the dataset is divided into 90% training set and 10 % test set.

Then the training set is divided into different folds. For this project $k = 10$ is used. Every split has a custom dataset with different validation and training sets. This means that a training set is divided into ten different splits of which every fold becomes the validation set once. The training set of a split is then assigned a boolean value that if True augments the image. Augmentation value, the chance that an image gets augmented, is 50%. This was discovered in previous research to gain the best performance [Seb23]. This serves to create a balance between overfitting and underfitting the training set. During this process the image data is also normalised to increase the efficiency of the model.

After this the splits are loaded via data loaders. The purpose of data loaders is to decouple the data samples from the model training code. It also serves as an easy way to pass the dataset in mini-batches [Pyta]. The data loaders in this case will always have a batch size of 16 which is because this worked best in previous research [Seb23]. These data loaders also shuffle the samples which enhances learning and makes the model more robust.

Now the model is moved to the GPU to accelerate the training significantly. The 3D CNN model's for every split's data loader will now train by first performing a forwards pass. This is so that predictions in the form of logits can be obtained. Then the loss can be calculated of the targets and logits using the categorical cross-entropy loss function. The categorical cross-entropy loss function is commonly used in neural networks with softmax activation in the output layer for multi-class classification tasks. By minimizing loss, the model learns to assign higher probabilities to the correct class while reducing the probabilities for incorrect classes, improving accuracy. It is a measure from the field of information theory, building upon entropy and generally calculating the difference between two probability distributions [Bro20]. The model has a learning rate of 0.0001 since this performed best for the ResNet3D model in previous research [Seb23]. The optimizer that will be used is AdamW. AdamW is a stochastic optimization method that is one of the most commonly used optimizers and it is a gradient descent optimization algorithm. It modifies the typical implementation of weight decay in Adam (Adaptive Moment Estimation), by decoupling weight decay from the gradient update [LH17]. It offers improved regularization and generalization performance. By incorporating weight decay directly into the optimization step, AdamW helps prevent overfitting and enhances model robustness. The optimizer is based on adaptive estimation of first-order and second-order moments. It is particularly useful in scenarios where there is a need for fast convergence and high precision. The models will evaluate itself through F1 score, accuracy and loss.

After all splits have trained for either 30 or 50 epochs, the averaged overall F1 score, accuracy and loss will be calculated on the testing set. This is done by moving the model to the GPU again and setting the model to evaluation mode. During training the overall time it took to train the model, the time it took for one epoch and the iterations per second will also be saved. After this a graph will be generated to showcase the performance of the model.

3.1 3D Image dataset

This paper will have a raw stack input of the 3d pollen grain images. This is the reason that a 3D convolutional neural network will be used. A 2D network is not able to capture the features that a 3D network is able to capture. The downside of this advantage is that it comes with the cost of more computational power. The reason for using the Z-stack instead of the technique that Chen li et al. used is that theoretically more information can be captured by the model. This could therefore lead to a possible improvement in the classification score. The dataset that will be used consists of 6472 images. They are made up of five species that are divided into three classes: Urtica (Urtica urens, Urtica dioica), Urtica membranacea and Parietaria (Parietaria judaica, Parietaria officinalis). They are divided this way because of the allergy levels that they cause [LPC+23]. The Urtica Membranacea is not originally from the Netherlands unlike the other species. This is an exotic Mediterranean species which is easily distinguished from the rest.

3.1.1 Whole Stack image

The whole stack image has the possibility to gain a better classification accuracy because of the spacial dependencies that can be captured over the frames in a pollen grain image. This can possibly allow for improved learning. It can however also give a problem because most frames are blurry and therefore may not contain any useful information.

3.1.2 Edge detection

Instead of the central subset of the stack that was used in the paper of Shreya [Seb23], this paper will focus on the frame with the best focal point. This focal point is not always in the middle and could therefore have lead to poor performance in the paper of Shreya. Therefore in this paper the central point of the Stack will be the layer that has the best focal point. To find this focal point a bit of preprocessing has been done. The focal point will be found through edge detection.

Edge detection is a term used to describe a significant local change in the image intensity [M22]. The pollen grain has a clearly visible edge which is best visible at the focal point. This means that the image with the highest intensity change is also the image with the best focal point. This makes the optimal frame easy to find and use as the central subsection. Different frame sizes will be tested in this paper to observe if more information will actually make a difference in classification accuracy. Figure 3 showed an example of how edge detection on a pollen grain looks like.

3.2 Preprocessing data

The dataset that will be used is the exact same as the one from Chen Li et al., with 6472 images [LPC+23]. Every image is in .tiff (Tagged Image File Format), which is a file format that is very

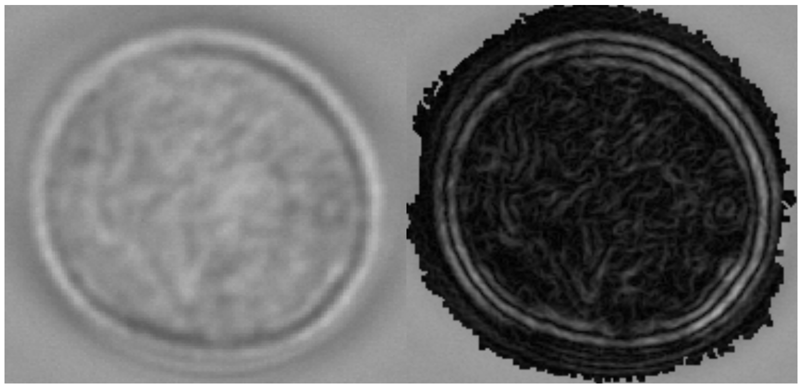

Figure 3: Example of edge detection on pollen grain. Figure on the left side is the normal one layer image of a pollen grain. On the right side is the same figure, but with the edges visualized.

flexible and is also lossless. The dataset needs to be preprocessed for the model to work with the model and to help in classification.

Edge detection as explained in section 3.1.2 is performed with the openCV library in python. For this project different optimal frame sizes will be tested. The difference between 4, 6, 8 and 20 frames will be tested for accuracy. These frames are based on the optimal frame that was found through edge detection. In some cases the optimal frame size was too close to the beginning or end of the image stack. This meant that if the surrounding frames were taken into consideration that they would not always have the same size. The model however needs to always have the same amount of frames to work. Therefore the optimal frames are chosen based on the position of the optimal frame in the Z-stack. If the optimal frame was too close to the edge of the Z-stack it was moved to a more centrally located frame. This means that if the optimal frame is in a location that is on the lower side of the stack that the optimal frame will also be on the lower side of the optimal frames chosen.

In order for the neural network to work all input data needs to have the same size. The images in the dataset do not all have the same size and therefore the images need padding. Padding is filling the edges of the image with a value of choice. For this project mean padding was implemented, thus the edges were filled with the average value of the image until the desired dimension was created. Mean padding was chosen instead of the zero padding that Shreya [Seb23] used in previous whole stack image research, because of the edge detection. Edge detection would detect a sudden switch in intensity which could negatively impact the neural network. The images were then also cropped so they would all have the same size. Cropping was done after the padding because some images had more than the desired 224x224 pixels, which could be fixed with cropping.

The train set for every split has been augmented to compensate for a small dataset size. The augmentation threshold that was chosen is 50%, as this is what was found to perform as discovered in previous research on the same dataset [Seb23]. The transformations that are applied for augmentation are random horizontal and vertical flips. This means that new artificial images are created that are flipped which can help with learning for the model.

3.3 Software Libraries

This project is built in python, which is because the 3D CNN models that are used require certain python libraries to function. The 3D CNN models are built through the Pytorch framework. The edge detection was performed through the openCV framework.

Pytorch is a python library made for deep learning. It is an open source tool with which you can make deep neural networks or you can use pre-existing models. The choice of Pytorch instead of Tensorflow like previous research [LPC+23] was made because of the 3D CNN models. Pytorch has ready-made 3D CNN models that can be imported and have the possibility of data augmentation using dataloaders. Pytorch also allows for GPU acceleration which is necessary for neural networks that are computationally very expensive.

The OpenCV python library has been used to perform edge detection. Open source computer vision library is the most used python library when is comes to computer vision problems. Through this library canny edge detection was used. This method greatly reduces the amount of data that needs to be processed [doc].

3.4 Models

Five different models were tested on the pollen dataset. These five models have three different Convolutional Neural Networks. These networks are ResNet3D, RetinaNet and Faster R-CNN. The RetinaNet and Faster R-CNN networks have been tested with two different backbones to create the five different models. These backbones are ResNet50 and ResNet3D. These object detection models have been chosen because of the better accuracy that was showcased in the paper [GGOGV+24] compared to the ResNet3D classification models. These models were all run with and without pre-trained weights.

3.4.1 Baseline ResNet3D

The ResNet3D classification model will serve as a baseline for the tests, since this model has already been tested on this dataset. This is the $r3d\_18$ model, which has been imported from the Pytorch library. It is a 3-dimensional 18-layer ResNet model. The pre-trained weights that were used are $R3D\_18\_Weights.KINETICS400\_V1$. These are the default weights from Pytorch [Pytb]. The weights were obtained by pretraining the model on two different datasets, the Kinetics [CZ17] and Sports-1M [KTS+14] dataset.

3.4.2 RetinaNet

RetinaNet is an object detection model that is good at classifying small and dense scale objects. It consists of a feature pyramid network (FPN) backbone on top of a feed forward ResNet architecture [GGOGV+24]. This network normally has an object detection layer however this has not been used for this paper since the dataset does not need this layer to find the pollen grains. The ResNet architectures used are ResNet50 and ResNet3D. A version of ResNet50 that has been used is from Facebook's Pytorchvideo [FMW+21]. This is a ResNet50 network that allows for 3D input. The ResNet50 model was chosen because of the very high accuracy score that Chen Li et al. was able to achieve with it [LPC+23], it was also the model that was used for the previous research with RetinaNet. The other network is ResNet3D, which is the network that is used for the baseline.

3.4.3 Faster R-CNN

Faster R-CNN is the other object detection model that was tested. This is a model that in comparison to RetinaNet has a bit higher inference times, however this is also the reason that it typically results in a higher mean precision [GGOGV+24]. The object detection layers from this network are also not utilised. The model uses the same backbones as the RetinaNet model.

3.5 Training

The training time for the neural networks is computationally very expensive. Therefore the models were trained on a NVIDIA RTX 4070Ti GPU. This is a GPU that has 12GB of VRAM. The time that it takes for a model to complete one epoch was measured with a python module called tqdm. This module is capable of showing a progress bar for how far the model is with training, and most importantly it showcases how many iteration per second the model can process, which is important information about how long training will take. The models were run with K-fold cross-validation with $k = 10$. Transfer learning was also used in this project. These techniques are explained in the subsections below.

3.5.1 K-Fold Cross-Validation

K-fold cross-validation is a robust method for assessing the performance of a predictive model, especially in scenarios where data is limited. The technique involves partitioning the training set into k smaller sets also called folds. The model is trained on $k - 1$ of these folds and validated on the remaining fold, ensuring that every fold serves as the validation set exactly once. This process is repeated k times, and the performance metric, in this case F1-score, is averaged over these iterations to provide a more robust estimate of the model's performance. This method mitigates the risk of overfitting that can arise from a single train-test split and efficiently utilizes the dataset [Sl].

3.5.2 Transfer learning

Transfer learning is a technique that essentially boils down to using a pre-trained model on a new dataset. It is a technique that can is very helpful for small datasets, because it uses the weights from a previous network to train the current network. Therefore less data is required to achieve the same accuracy that a non pre-trained model requires. This technique also helps with the reduction of training time [Don22]. Transfer learning in this project was implemented with pre-trained weights and using a previous iteration of the model.

3.6 Testing

After all splits are done with training a test function is used. This function will test the 3D CNN model for every split on a train set which it has not seen before. These result are then averaged and saved. This does not require a GPU since testing does not need as much computational power compared to training.

4 Results

The results for all 5 different models will be discussed in the three section below. The section are in the following order: ResNet3D, Faster R-CNN and then RetinaNet.

4.1 Baseline ResNet3D

The ResNet3D baseline was first tested on performance comparing pre-trained and non pre-trained models. These tests were conducted with 6 optimal frames and 30 epochs.

4.1.1 Pre-trained model

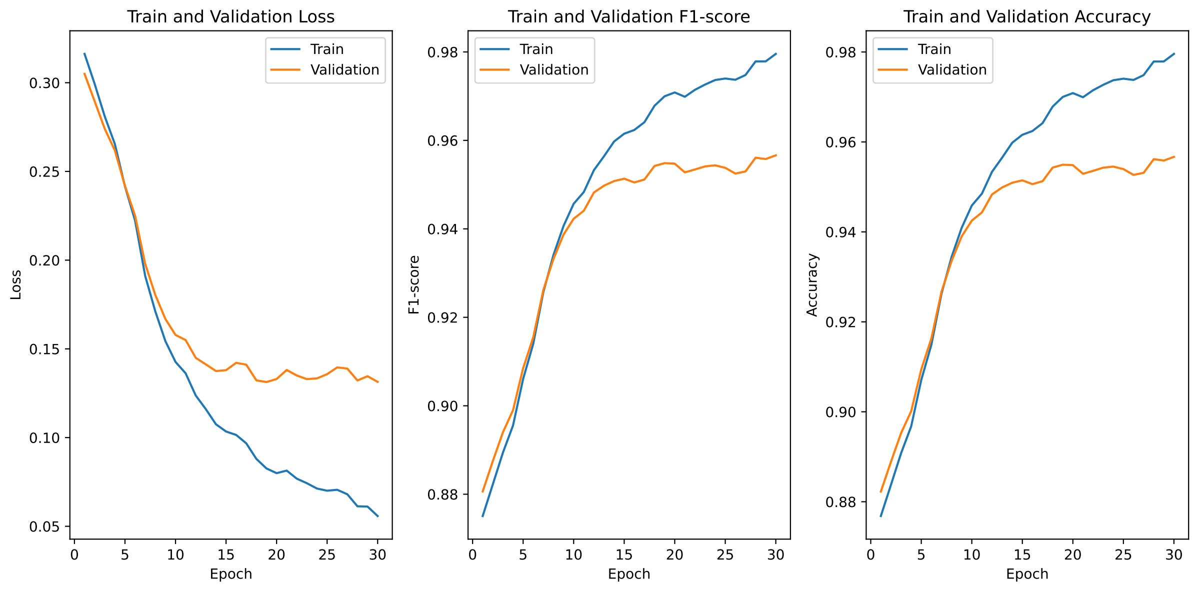

Figure 4: These graphs represent the ResNet3D model. On the test set this model achieved a F1-score of 95.9%, accuracy score of 95.9% and a loss of 0.112. This model was not pre-trained.

Figure 4 showcases the model that is not pre-trained. The F1-score on this model is already better compared to the previous model made in previous research of whole stack 3D data, in that research the model was able to achieve a score of 95% after hyperparameter optimization [Seb23].

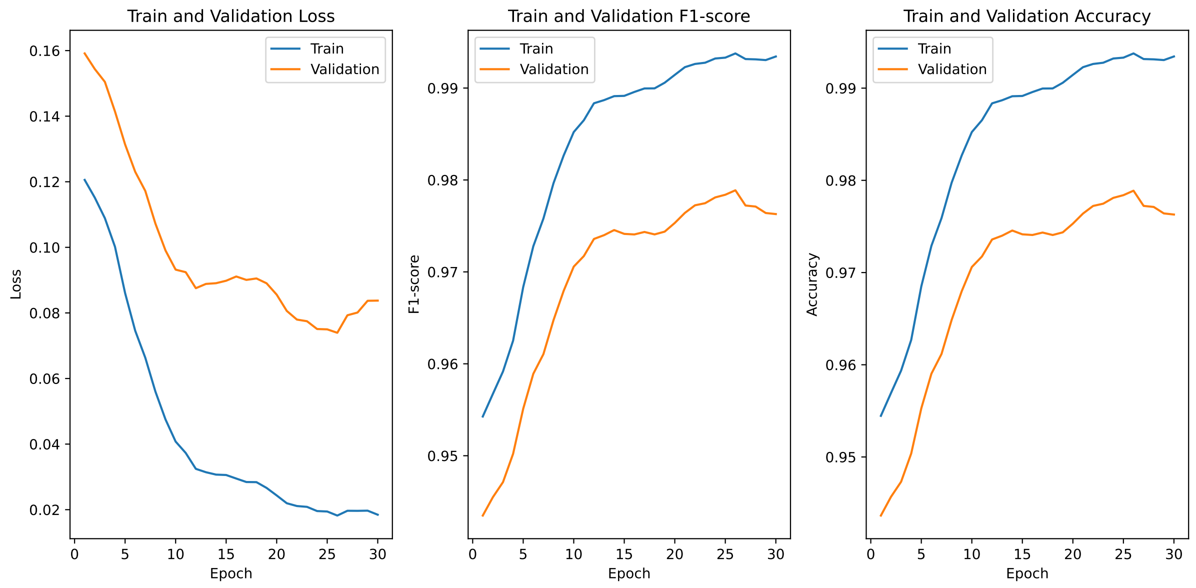

Figure 5: These graphs represent the pre-trained ResNet3D model with 6 optimal frames. On the test set this model achieved a F1-score of 98.1%, accuracy score of 98.1% and a loss of 0.052.

Figure 5 showed that the pre-trained model compared to the non pre-trained model performs significantly better with a F1-score of 98.1%. The time it took for both models to train was also almost the same, since they both took 92 seconds per epoch to process the test set. Therefore for the rest of the tests of ResNet3D models a pre-trained model will be used.

ResNet3D	loss	F1-score	accuracy	Time for one epoch (in seconds)
Non pre-trained	0.112	0.959	0.959	92
Pre-trained	0.052	0.981	0.981	92

Table 1: ResNet3D model testing performance between pre-trained and non pre-trained model

4.1.2 Optimal frame selection

The optimal frames were tested to observe if increasing the amount of frames makes a difference to the accuracy of the model. This is to test if a subset of a frame that has the best focal point in the middle makes a difference compared to the previously tested frame subsections.

Figure 10 showed a decline in accuracy around epoch 15 and in between epochs 25 and 30. This sudden decline is also slightly visible in figure 5. This decline is however not visible in figures 11, 12 and 13 which can be found in the appendix.

The results, summarized in table 2, showed that the difference between subsets of frames do not indicate that a significant better performance can be gained by increasing the amount of frames. Increasing the amount of frames only resulted in a 0.02 increase of accuracy. The sudden decline in accuracy and the fact that the models do not stagnate give a reason to test the model with more epochs. The rest of the research was performed with 6 optimal frames instead of the 10 frames, because of the extra time that it takes for the model to train with these 4 extra frames. The extra training time results in 30 minutes extra of training when run in K-fold. This does not seem like much, however when run with 50 epochs this difference suddenly becomes a lot bigger.

Frames	loss	F1-score	accuracy	Time for one epoch (in seconds)
4 frames	0.055	0.980	0.980	90
6 frames	0.052	0.981	0.981	92
8 frames	0.050	0.980	0.980	92
10 frames	0.046	0.983	0.983	93
20 frames	0.069	0.977	0.977	97

Table 2: ResNet3D model testing performance between pre-trained and non pre-trained model

4.1.3 Increasing amount of epochs

To test if the learning curves from ResNet3D do not stagnate, 50 epochs of the model with 6 optimal frames was run.

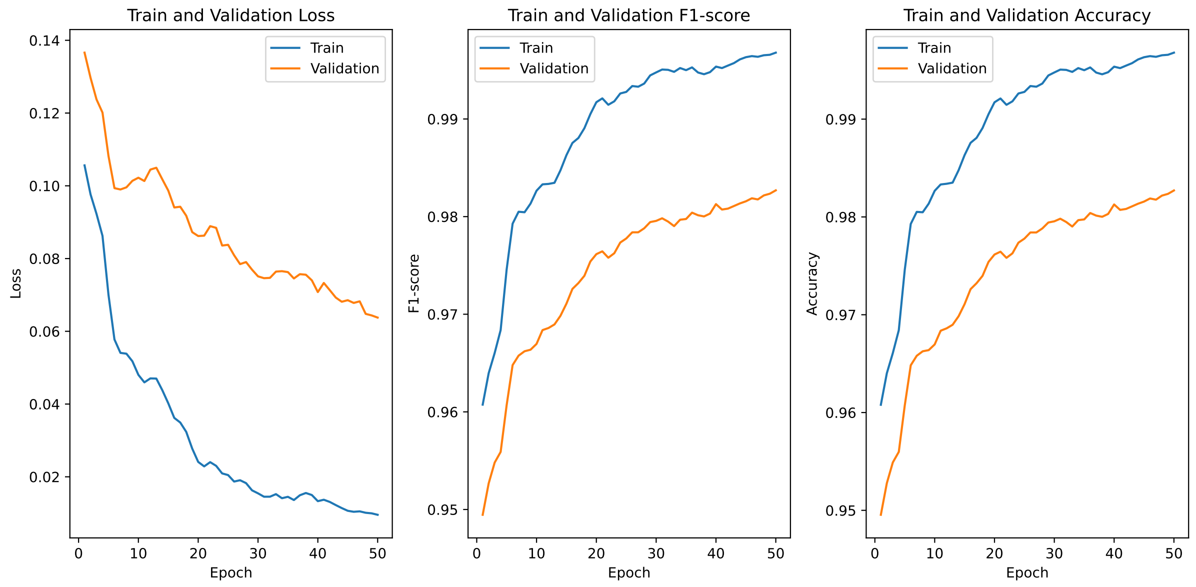

Figure 6: These graphs represent the pre-trained ResNet3D model with 6 optimal frames and 50 epochs. On the test set this model achieved a F1-score of 98.2%, accuracy score of 98.2% and a loss of 0.053.

From figure 6 the conclusion can be drawn that an increase in accuracy is only increasing the accuracy marginally with 0.01 for 20 epochs extra. This can also be seen in table 3. The very small increase in accuracy does not make the trade off with 20 extra epochs seem very attractive.

ResNet3D	loss	F1-score	accuracy	Time for one epoch (in seconds)
30 epochs	0.052	0.981	0.981	92
50 epochs	0.053	0.982	0.982	92

Table 3: ResNet3D model testing performance difference between 30 and 50 epochs

4.2 Faster R-CNN

Then the Faster R-CNN model was tested. The model was first run with the ResNet50 backbone.

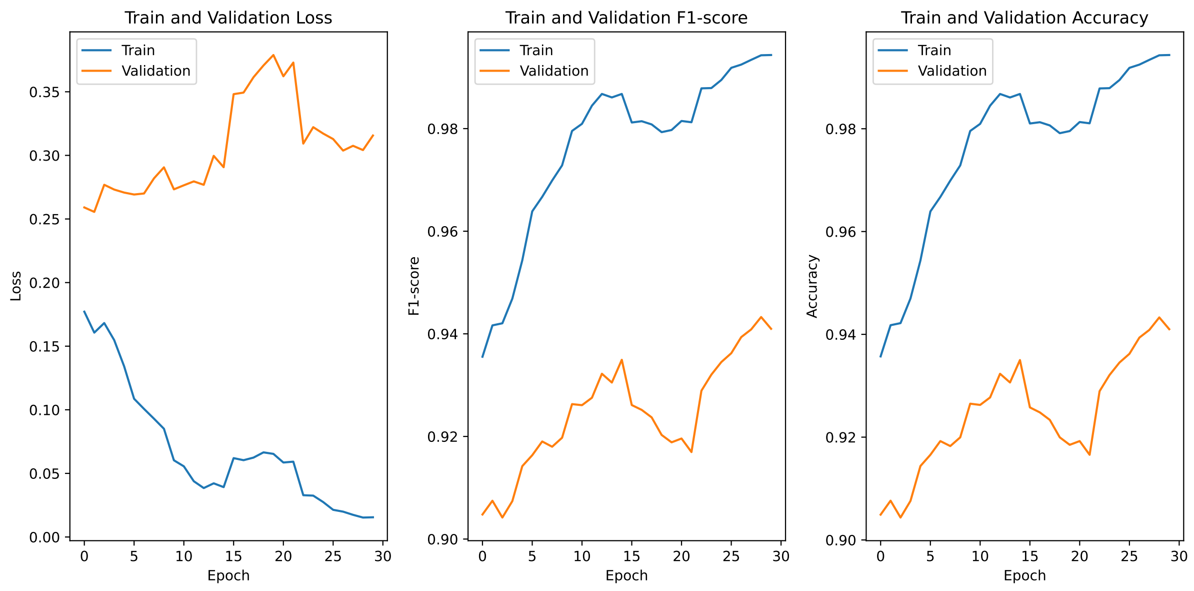

Figure 7: These graphs represent non pre-trained Faster R-CNN model with ResNet50 backbone. On the test set this model achieved a F1-score of 94.3%. This was also the only run without K-fold cross-validation.

Figure 7 is the result of the model, this test was not run in K-Fold. This was done because of the time it took to test just one of the ten splits. It took more than five hours to train just one split, because one epoch would take 10 minutes to train. The results from this figure also indicate that the model still needs a lot of epochs before the results get close to the performance of ResNet3D. The model was only able to achieve an F1-score of 94.3% on the test set after 30 epochs. Therefore this model is not that useful for the classification task.

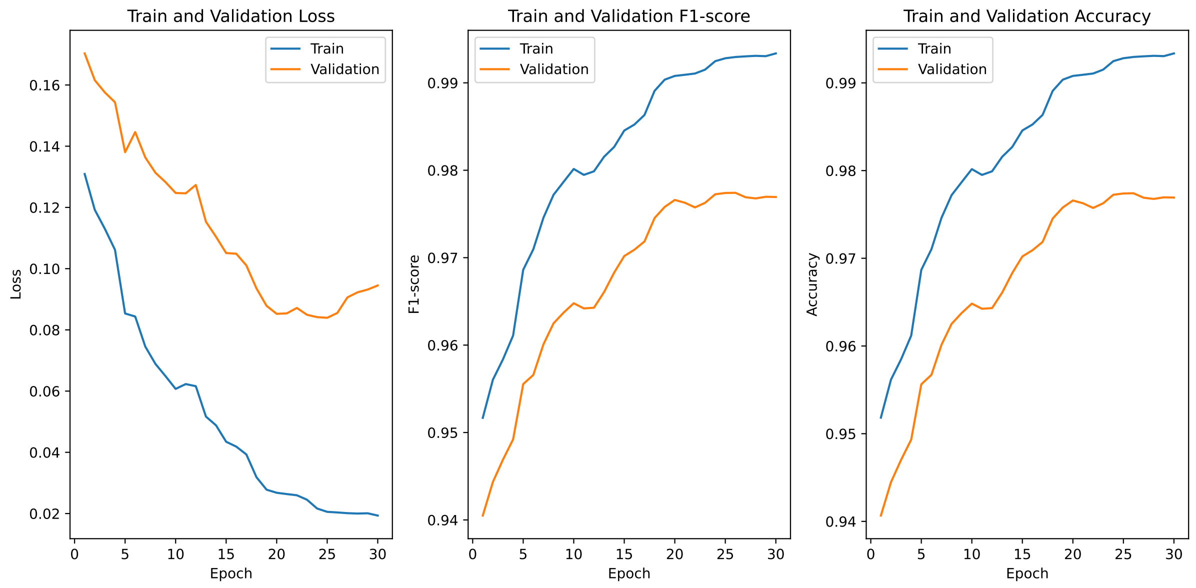

Figure 8: This graph represent the pre-trained Faster R-CNN model with ResNet3D backbone. On the test set this model achieved a F1-score of 97.9%, accuracy score of 97.9% and a loss of 0.072.

Figure 8 showed that the Faster R-CNN model with ResNet3D backbone is a lot better compared to the ResNet50 backbone version. The model achieved a F1-score of 97.9%, an accuracy of 97.9% and a loss of 0.072. The model took 115 seconds per epoch to train.

4.3 RetinaNet

The last model to be tested was RetinaNet. First a pre-trained ResNet50 backbone model was compared to a non pre-trained model. Figure 14 and 15 showed the performance of a non pre-trained and pre-trained model. Both of the models stagnate after 20 epochs, however the pre-trained model has a significantly higher F1-score.

Then it was tested if an increase in the total epochs would make a difference for the ResNet50 backbone. Figure 16 shows the graphs for 50 epochs, figure 15 for 30 epochs. The only difference that increasing the epochs has on the model, is that the loss decreases. Therefore there is no use for the model to be trained with 50 epochs. The last test that was run is the comparison between the different backbones for RetinaNet. Unlike the Faster R-CNN model this ResNet50 model was a lot faster. Therefore a comparison can be made.

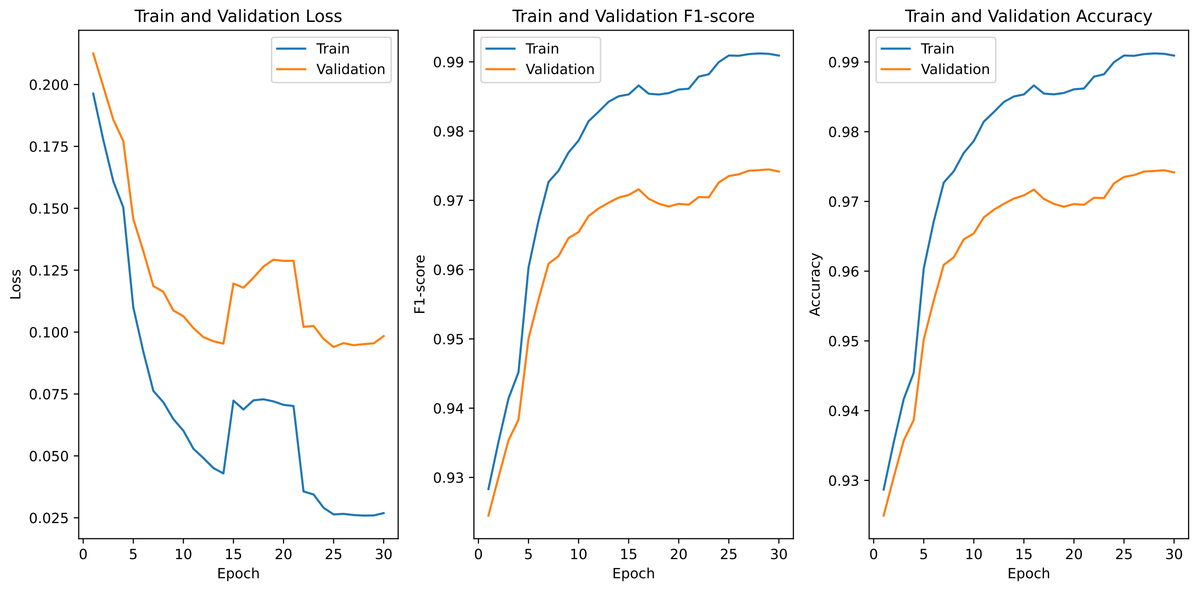

Figure 9: These graphs represent the pre-trained RetinaNet with ResNet3D backbone that was trained on 30 epochs. On the test set this model achieved a F1-score of 97.8%, accuracy score of 97.8% and a loss of 0.060.

Figure 9 showed the performance of the ResNet3D backbone. This backbone performs significantly better compared to the ResNet50 backbone with a F1-Score of 97.8%. Table 4 showed the results of the RetinaNet experiments. From this it can be concluded that The ResNet3D backbone performance better compared to ResNet50.

RetinaNet backbone	epochs	loss	F1-score	accuracy	Time for one epoch (in seconds)
ResNet50 non pre-trained	30	0.151	0.954	0.954	100
ResNet50	30	0.134	0.965	0.965	99
ResNet50	50	0.053	0.965	0.965	99
ResNet3D	30	0.060	0.978	0.978	115

Table 4: Results of performance RetinaNet models

5 Conclusions

The primary objective of this study was to enhance the classification accuracy of Urticaceae pollen using deep learning techniques, specifically leveraging 3D convolutional neural networks (3D CNNs). This was motivated by the need for precise identification of pollen grains due to their varying allergenic potential, which is crucial for environmental and health monitoring.

The research focused on comparing different deep learning models, including a baseline ResNet3D, Faster R-CNN, and RetinaNet, with particular attention to the performance of the ResNet50 and

ResNet3D backbones for the object detection models. The models were trained and tested on a dataset consisting of 6472 pollen stack images, each captured with 20 slices along the Z-axis.

The ResNet3D backbone demonstrated superior performance compared to the ResNet50 backbone, achieving an F1-score and accuracy of 97.8% on the test set with RetinaNet. However the baseline ResNet3D classification model performed better compared to the object detection and classification models. The baseline Resnet3D achieved an F1-score and accuracy of 98.3% when trained with 10 optimal frames. This indicates the efficacy of 3D CNNs in capturing spatial features from stack images, which is critical for distinguishing between morphologically similar pollen grains.

Incremental improvements in model performance were observed with the optimization of frame selection and an increase in the number of epochs. However, the enhancements were marginal, suggesting that the models had already achieved near-optimal performance under the given conditions.

The implementation of edge detection to identify the best focal point significantly contributed to the improved classification accuracy. By focusing on the image layer with the highest intensity change, the model could utilize the most informative features of the pollen grains. This preprocessing step mitigated the issue of blurry frames and ensured that the input to the neural network was of the highest quality, thereby enhancing the model's ability to learn and distinguish between different pollen types.

The near parity of the F1 score and accuracy in our models is particularly significant. This indicates a balanced performance across both precision and recall, suggesting that the model is equally effective at correctly identifying true positives and minimizing false positives. Such balance is crucial in applications like pollen classification, where both the presence of pollen and the correct identification of its type are important for accurate environmental monitoring and allergen forecasting.

Despite the advancements achieved in this study, the previous research by Chen Li et al. [LPC+23] still demonstrated slightly better performance. Their approach, which utilized simple image processing techniques combined with various deep learning architectures, achieved an accuracy of 99.4% in classifying Urticaceae pollen. This underscores the potential for further optimization in our methods, possibly by integrating additional preprocessing techniques or refining the neural network architectures to match or surpass their results.

6 Future Work

This research has a few possibilities to work on in the future. There were a few obstacles during this project that could use further research.

Currently the optimal focal point selection is not always entirely correct. If an optimal focal point frame is near one of top or bottom layers and the total amount of subset frames is set to high, it might displace the middle frame to a point more centralized. This could have an impact on the 3D CNN's, and therefore further research could look into the impact of using a dataset with more slices along the Z-axis.

In possible future research the scope of pollen classification could be expanded to include a wider variety of pollen families beyond Urticaceae. Investigating the application of 3D CNN's to other pollen types could validate the generalizability and robustness of the proposed methodology. This expansion would involve collecting and annotating new datasets from different plant families, which may present unique challenges in terms of morphological diversity and data complexity. Additionally, understanding the specific allergenic properties of various pollen families would further enhance the practical applications of these models in environmental health and allergen forecasting.

References

[AGM+20] Gilberto Astolfi, Ariadne Barbosa Gonçalves, Geazy Vilharva Menezes, Felipe Silveira Brito Borges, Angelica Christina Melo Nunes Astolfi, Edson Takashi Matsubara, Marco Alvarez, and Hemerson Pistori. POLLEN73S: An image dataset for pollen grains classification. *Ecological informatics*, 60:101165, 11 2020.

[Bro20] Jason Brownlee, PhD. A gentle introduction to Cross-Entropy for Machine Learning, 12 2020.

[BWT+19] T. Biedermann, L. Winther, S. J. Till, P. Panzner, A. Knulst, and E. Valovirta. Birch pollen allergy in europe. *Allergy*, 74(7):1237–1248, 2019.

[CY09] Clayton M. Costa and Suann Yang. Counting pollen grains using readily available, free image processing and analysis software. *Annals of Botany*, 104(5):1005–1010, 07 2009.

[CZ17] Joao Carreira and Andrew Zisserman. Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset. *arXiv (Cornell University)*, 1 2017.

[DCNMO+20] Gennaro D'Amato, Herberto Jose Chong-Neto, Olga Patricia Monge Ortega, Carolina Vitale, Ignacio Ansotegui, Nelson Rosario, Tari Haahtela, Carmen Galan, Ruby Pawankar, Margarita Murrieta-Aguttes, Lorenzo Cecchi, Christian Bergmann, Erminia Ridolo, German Ramon, Sandra Gonzalez Diaz, Maria D'Amato, and Isabella Annesi-Maesano. The effects of climate change on respiratory allergy and asthma induced by pollen and mold allergens. *Allergy*, 75(9):2219–2228, 2020.

[doc] OpenCV docs. OpenCV: Canny Edge Detection.

[Don22] Niklas Donges. What Is Transfer Learning? Exploring the Popular Deep Learning Approach., 9 2022.

[DRB16] Amar Daood, Eraldo Ribeiro, and Mark Bush. Pollen grain recognition using deep learning. In George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Fatih Porikli, Sandra Skaff, Alireza Entezari, Jianyuan Min, Daisuke Iwai, Amela Sadagic, Carlos Scheidegger, and Tobias Isenberg, editors, *Advances in Visual Computing*, pages 321–330, Cham, 2016. Springer International Publishing.

[FMW+21] Haoqi Fan, Tullie Murrell, Heng Wang, Kalyan Vasudev Alwala, Yanghao Li, Yilei Li, Bo Xiong, Nikhila Ravi, Meng Li, Haichuan Yang, Jitendra Malik, Ross Girshick, Matt Feiszli, Aaron Adcock, Wan-Yen Lo, and Christoph Feichtenhofer. PyTorchVideo: A deep learning library for video understanding. In *Proceedings of the 29th ACM International Conference on Multimedia*, 2021. https://pytorchvideo.org/.

[GGOGV+24] Ramón Gallardo, Carlos J. García-Orellana, Horacio M. González-Velasco, Antonio García-Manso, Rafael Tormo-Molina, Miguel Macías-Macías, and Eugenio Abengózar. Automated multifocus pollen detection using deep learning. *Multimedia tools and applications*, 2 2024.

[Gup24] Ayush Gupta. A Comprehensive Guide on Optimizers in Deep Learning, 7 2024.

[IBM] IBM. What are Convolutional Neural Networks? — IBM.

[Kei23] Zoumana Keita. An Introduction to Convolutional Neural Networks (CNNs), 11 2023.

[KTS+14] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[LH17] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv (Cornell University)*, 1 2017.

[LPC+23] Chen Li, Marcel Polling, Lu Cao, Barbara Gravendeel, and Fons J. Verbeek. Analysis of automatic image classification methods for Urticaceae pollen classification. *Neurocomputing*, 522:181–193, 2 2023.

[M22] Vikram M. Comprehensive Guide to Edge Detection Algorithms, 8 2022.

[Opp22] Artem Oppermann. How Loss Functions Work in Neural Networks and Deep Learning, 12 2022.

[Pyta] Pytorch. Datasets DataLoaders — PyTorch Tutorials 2.3.0+cu121 documentation.

[Pytb] Pytorch. r3d_18 — Torchvision main documentation.

[RBD+23] Masoud A. Rostami, Behnaz Balmaki, Lee A. Dyer, Julie M. Allen, Mohamed F. Sallam, and Fabrizio Frontalini. Efficient pollen grain classification using pre-trained Convolutional Neural Networks: a comprehensive study. *Journal of big data*, 10(1), 10 2023.

[Seb23] Shreya Sebastian. Enhancing pollen classification accuracy through the use of whole stack images in 3D convolutional neural networks, 2023.

[Sl] Scikit-learn. 3.1. Cross-validation: evaluating estimator performance.

[Ten] Tensorflow. Video classification with a 3D convolutional neural network.

7 Appendix

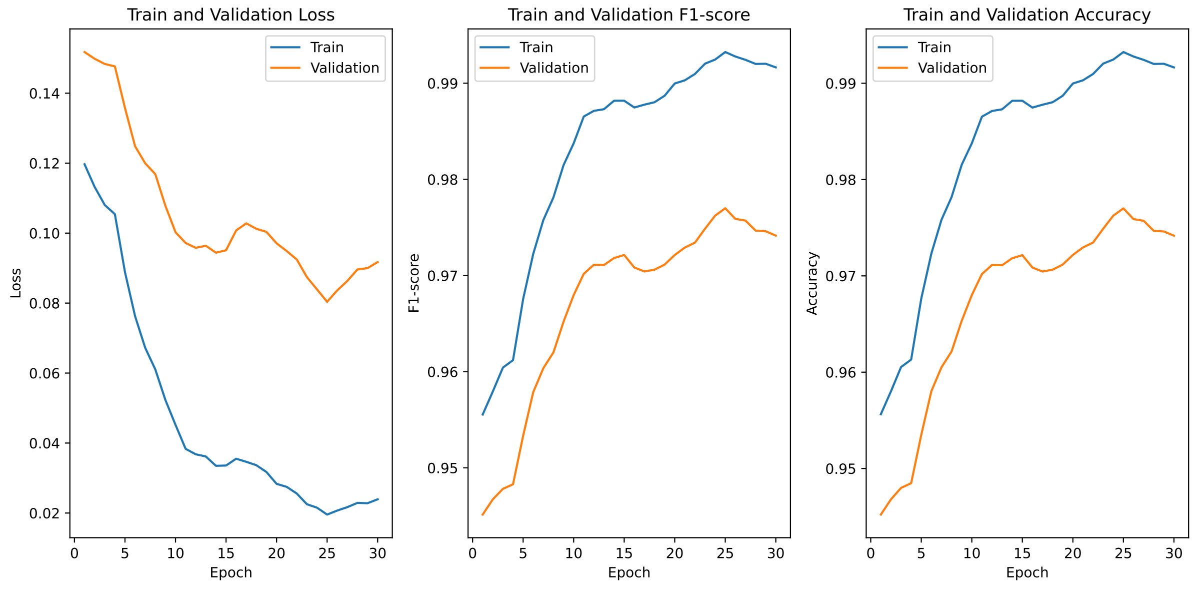

Figure 10: These graphs represent the pre-trained ResNet3D model with 4 optimal frames. On the test set this model achieved a F1-score of 98.0%, accuracy score of 98.0% and a loss of 0.055.

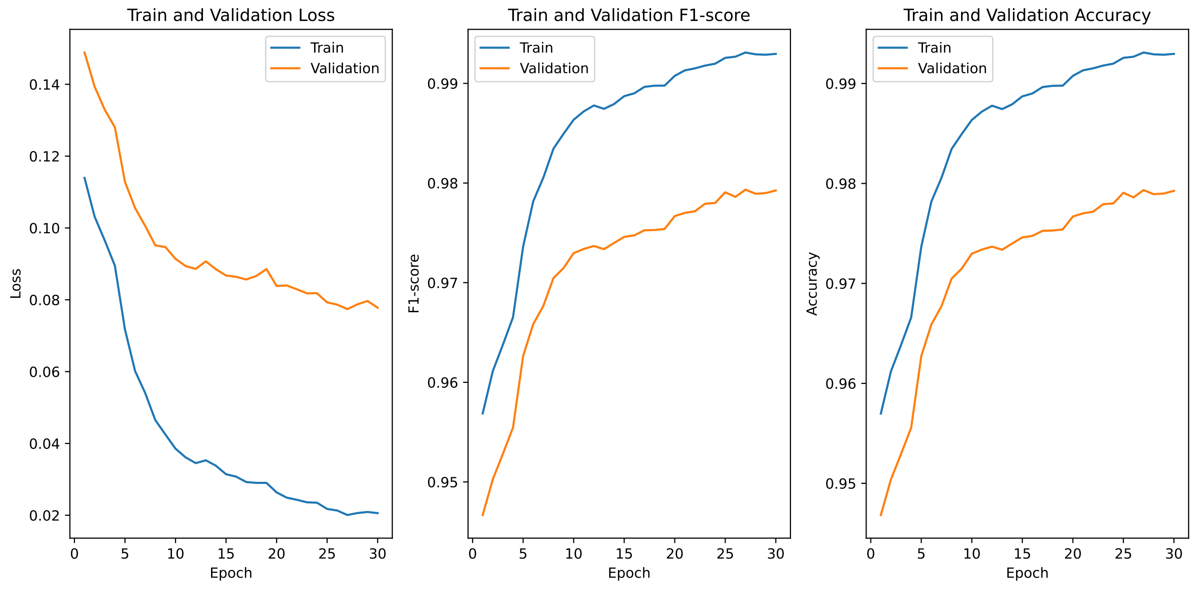

Figure 11: These graphs represent the pre-trained ResNet3D model with 8 optimal frames. On the test set this model achieved a F1-score of 98.0%, accuracy score of 98.0% and a loss of 0.050.

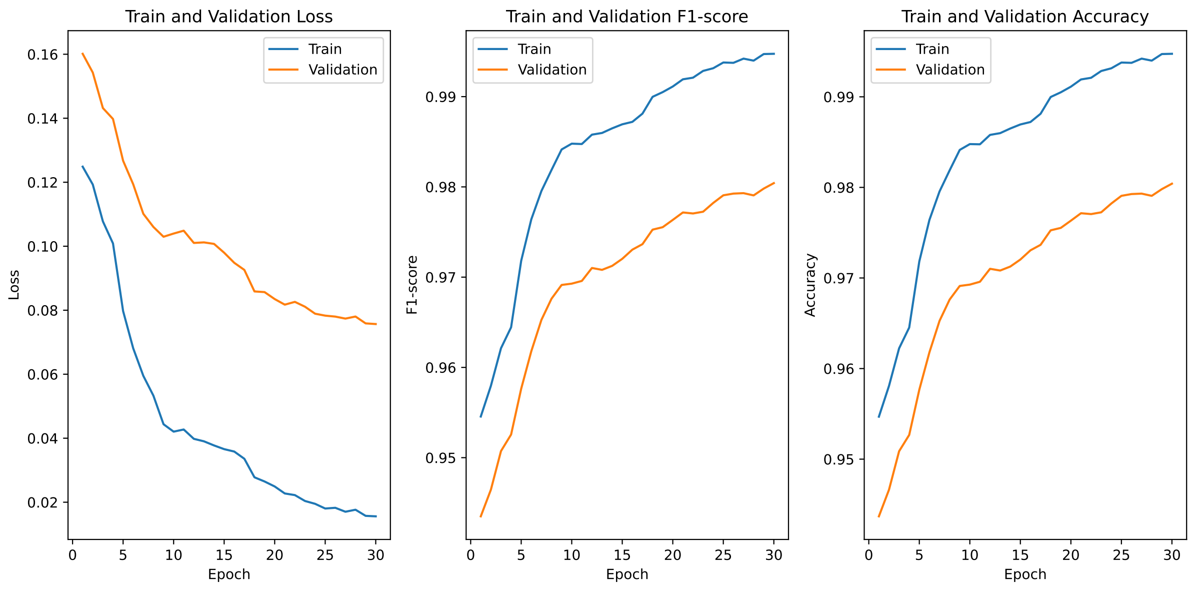

Figure 12: These graphs represent the pre-trained ResNet3D model with 10 optimal frames. On the test set this model achieved a F1-score of 98.3%, accuracy score of 98.3% and a loss of 0.046.

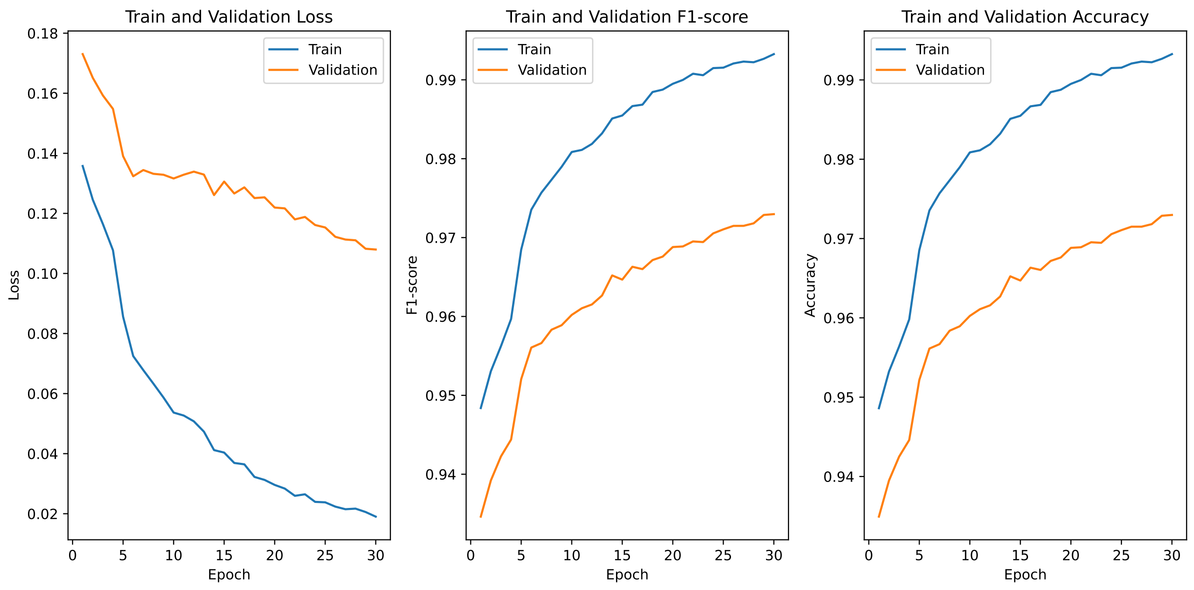

Figure 13: These graphs represent the pre-trained ResNet3D model with 20 optimal frames. On the test set this model achieved a F1-score of 97.7%, accuracy score of 97.7% and a loss of 0.069.

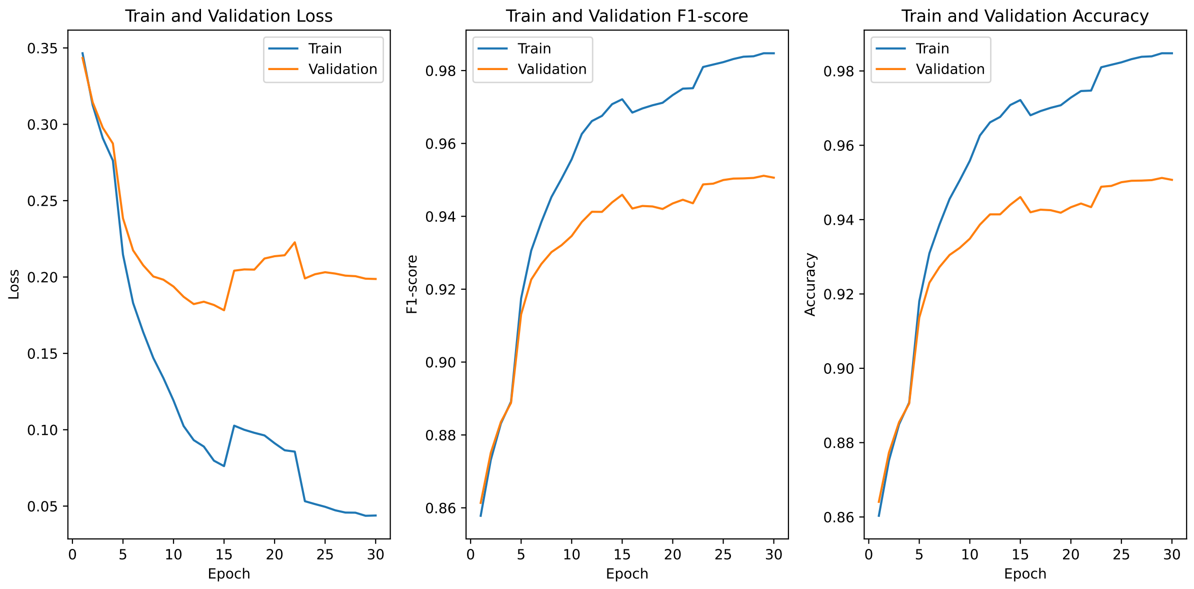

Figure 14: These graphs represent the non pre-trained RetinaNet model with ResNet50 backbone that was trained on 30 epochs. On the test set this model achieved a F1-score of 95.4%, accuracy score of 95.4% and a loss of 0.151.

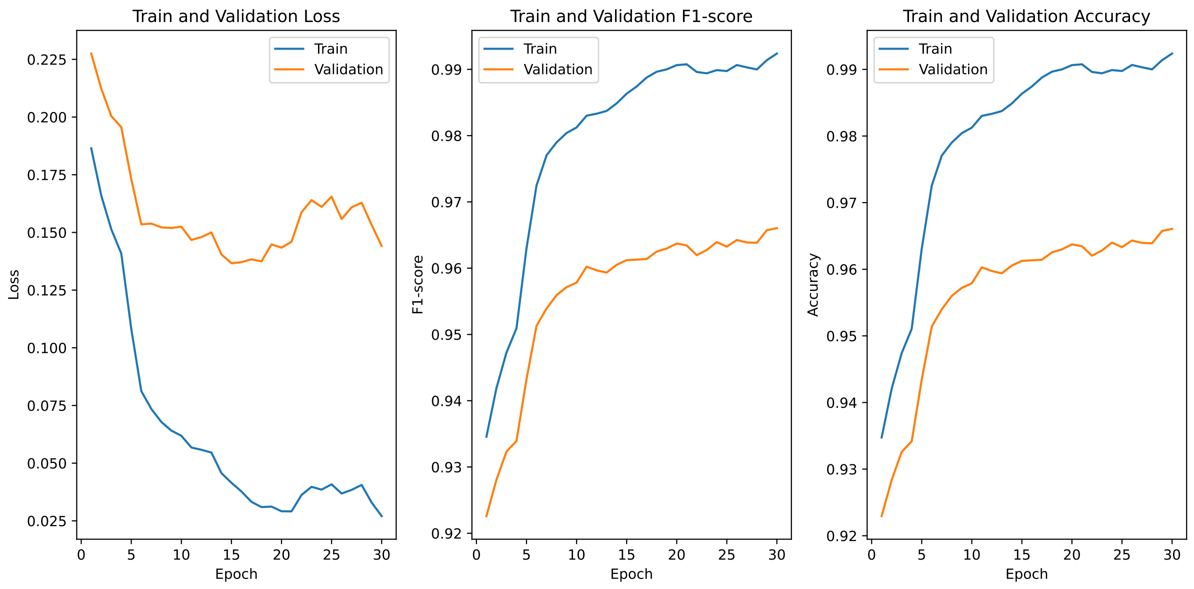

Figure 15: These graphs represent the pre-trained RetinaNet with ResNet50 backbone that was trained on 30 epochs. On the test set this model achieved a F1-score of 96.5%, accuracy score of 96.5% and a loss of 0.134.

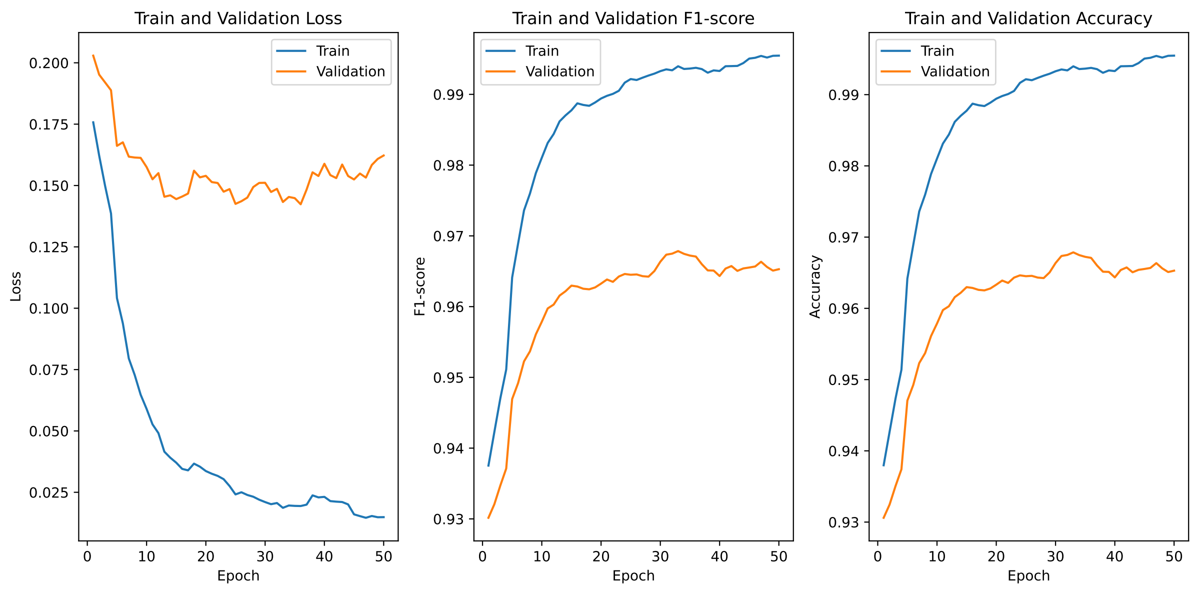

Figure 16: These graphs represent the pre-trained RetinaNet with ResNet50 backbone. On the test set this model achieved a F1-score of 96.5%, accuracy score of 96.5% and a loss of 0.148. This model has trained for 50 epochs