''

Universiteit Leiden
The Netherlands

Opleiding Informatica

Measuring the performance of different convolutional neural networks and a Swin-transformer model in pollen classification using 3D sequential data

Imaan Bijl

Supervisors:
Lu Cao (first reader) & Fons Verbeek (second reader))

BACHELOR THESIS CONCEPT 1

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

24/7/2024

Abstract

Human-induced climate change will result in an increased amount of allergenic airborne pollen. The allergenic properties of different plant species vary. Therefore, a correlation between the amount and type of pollen in the air and the amount and severity of symptoms for hay fever patients could prove relevant and be investigated. This paper attempts to maximize performance on pollen classification using three different models: two convolutional neural networks (ResNet and MobileNetV2) and one transformer model (Swin Transformer). The data used for training and testing the models is processed in 3D form, which is unusual in pollen classification; most other papers in this field use a 2D projection derived from the original 3D data. The two convolutional neural networks outperform the transformer model when the pre-trained models are deployed to the dataset. Accordingly, the three models were edited to be able to process 3D data. The ResNet model yields a maximum accuracy of 98.9 percent, and the MobileNetV2 model, which is much quicker and more lightweight, yields a maximum accuracy of 97.4 percent.

Contents

1 Introduction

1.1 Biological and pathological background

Pollen grains are the male reproductive cells of seed plants. Pollen can be transported by insects or other animals, or through air. A single pollen grain contains enough genetic information to create a new plant ['Po21]. Humans may experience an allergenic reaction when exposed to airborne pollen. This phenomenon is called hay fever (or allergic rhinitis). Different patients may experience sensitivity to different pollen grains. While symptoms can be mild for some patients, by the number of patients, hay fever is among the non-fatal diseases placing the greatest burden on public health [SCH16]. Since different pollen species possess different allergenic properties, identifying which species are present in the air tells a lot about the extent to which hay fever patients will experience symptoms [LPC+23]. Pollen classification is an important task in various medical and biological branches [BOT+20]. Previously, it was mainly done by hand and required trained experts. It is a tedious and time-consuming process since the pollen grains of different plant species may show only subtle morphological differences [LPC+23]. By utilizing machine learning and deep learning techniques, the costs of pollen classification can be reduced and the need for a trained specialist can be eliminated. This is especially needed since climate change and increased urbanization are expected to cause a rise in the concentration of allergenic pollen present in the air [DCNMO+20].

Two strains of the Urticaceae plant family, namely Urtica and Parietaria, produce morphologically similar pollen grains. Even experts struggle to tell apart the pollen grains of the two strains. Only the pollen of the species Urtica membranacea can be distinguished with relative ease.
However, there is a significant difference in allergenic properties between Urtica and Parietaria. While hay fever patients notice little from the pollen grains of the Urtica strain, the Parietaria, a strain more common in the Mediterranean, causes a severe allergic reaction. [CPIM18]. In this paper, classification is performed between Urtica and Parietaria strains. A separation is made between three groups: pollen from Parietaria Judaica (Pellitory of the Wall) and Parietaria Officinalis (eastern Pellitory of the Wall), Urtica Dioica (common stinging nettle), and Urtica Urens (dwarf stinging nettle) and lastly Urtica membranacea (large leafed nettle).

1.2 Literature Review

In 2024, Baokai Zu et.al published a paper on a Swintrainformer model's performance on Pollen classification [ZCL+24]. Here, the researchers proposed a new neural network for assigning images of pollen to the correct species. The network was based on the Swin transformer network architecture. This model type has been shown to outperform the classic convolutional neural networks on various computer vision tasks, especially those dealing with sequential data [ZCL+24]. Zu et al. tested the performance of their model on two separate data sets, one they made themselves, consisting of many Chinese species of pollen, and one public dataset representative of the pollen of European species. The data consisted of 2D microscopic pollen images in color, most of which were blurry. To combat this issue of blurry data, the researchers first fed the data through a so-called Enhanced Super-Resolution Transformer (ESRT), which improved the image's resolution by applying shallow and deep feature extraction, followed by image reconstruction. The result of this step is a high-resolution colored 2D image. This was also the shape in which the data was eventually fed

into the Swin-transformer model. When using the models on pre-trained weight, they achieved an accuracy score of 0.996 and an F-1 score of 0.9995 with their SwinT-ESRT model and 0.992 with a regular Swin-transformer model. These scores exceeded the ones of the other models used in the experiment: ViT, F2T-ViT, Efficient-NetV2, ConvNeXt, ResNet-50, and ResNet-34 [ZCL+24].

In their paper Huang et al. compare the performance of a 3d swin-transformer model to other state-of-the-art image classifiers: DINO, MoCo v3, MOBY and SiT on various tasks concerning commonly used datasets containing 3D or sequential data. The 3d Swin-transformer model performs better than all other models with a margin of 1,3 percent in the average accuracy. While this paper does not concern pollen classification, interestingly, the researchers attribute this strong performance of the 3D Swintranformer model to its ability to extract information from complex and variable multiscale features in 3D data, which are also found in the dataset of this thesis [HDLG22].

The paper by Li et al. titled *Analysis of automatic image classification methods for Urticaceae pollen classification"* discusses the efficiency of both machine learning and deep learning-based methods for classifying Parietaria and Urtica pollen strains. In the paper, a comparison is made between deep learning methods and machine learning methods on pollen classification [LPC+23]. The machine learning methods required the researchers to manually select important features of the pollen before they can be extracted from the individual pollen images. These features were selected based on the observable biological differences between the plant species. The deep learning methods do not require such feature selection. The image acquisition workflow consisted of taking the image of the pollen cells from 20 different focal depths. Next, three different projections are created along the Z-axis: the standard deviation projection, the minimum intensity projection, and the maximum intensity projection. The three different projections were fed into the models as separate channels. The deep learning methods generally outperformed the machine learning methods. The highest-scoring deep-learning method was ResNet50 (0.994 accuracy on 10-fold cross-validation). This model was closely followed by VGG19 (0.986 accuracy on tenfold cross-validation) and MobileNetV2 (0.985 on tenfold cross-validation)[LPC+23].

The same Z-stack projection was used in the paper of Polling et al. from May 2021, where the performance of several convolutional neural nets on classification was examined, using the microscopic images of pollen, that could not be manually differentiated beyond the genus level by specialists. Among the strongest performing models at the task were MobileNetV2 as well as VGG19 (accuracies of respectively 98.30 and 98.45 on tenfold cross-validation)[PLC+21].

Daood, connected to the Florida Institute of Technology proposes several different methods of pollen classification in his paper Pollen Grains Recognition Based on Computer Vision Methods. The most successful method entails the combination of a convolutional neural network called VGG16 and a customized recurrent neural network for the classification of multi-focal images, represented in a Z-stack. When combining this with transfer learning Daood achieved a perfect accuracy score [Dao18]

1.3 Data Handling

When looking at the papers described in subsection 1.2, it is clear that while several different pollen datasets exist, there are some distinct ways in which the data is usually fed into the models. Most commonly, the data is in 2D form. Some papers extract information from the 2D data by making projections of the image or from the stack of images from different focal points (most commonly MIN, MAX, and STD). In this paper, however, the raw stack of six frames is the input of the model. There will be no pre-processing in the form of making projections. The idea behind this is that minimal information is lost this way. Additionally, the sequence in which the stacks are ordered may contain information that would be lost when compressing the stack into 2D form.

1.4 Overview

In this paper, the performance of two different convolutional neural networks and one transformer-based network will be evaluated on the task of pollen classification. The classification made will be between pollen grains of the species mentioned in 1.2. The data is treated as sequential data, and after pre-processing, the Z-stack itself will be the input to the models, instead of a projection. The Swintransformer model is known to be good at temporal or sequential data. The MobileNetV2 lends itself well for tasks requiring a lot of computational power, because of its small size and few trainable parameters. K-fold cross-validation will be used for training and validation on the training data. The model's performance will be assessed based on its accuracy score on the test set.

2 Relevant Techniques and Models

2.1 Transfer Learning

Transfer learning is a technique widely used in computer vision where the starting point of training a model for a specific task A is the model trained on a different task B. One takes the model trained for task B and trains it on the existing training data for task A. The weights learned by training on task B are transferred to the next task A, hence the name transfer learning. Since a model's performance is generally related to the amount and quality of training data available, transfer learning can be of great aid when the available training data is small or of bad quality [Boond]. Two of the models discussed in this paper, namely MobileNetV2 and ResNet, are both tested when trained from scratch and when pre-trained. Unfortunately, there were no pre-trained weights to be found for the Swintransformer model. This model is therefore only tested when trained from scratch. Both MobileNetV2 and ResNet are trained on the ImageNet dataset in this paper. The ImageNet dataset contains over 14 million high-resolution labeled images with a wide variety of classes. [DDS$^+$09]

2.2 Tenfold cross-validation

K-fold cross-validation is a statistical method to assess a model's performance on a task. Its goal is to produce an accuracy measure that is reliable and robust. It does so by partitioning the data into k different subsets of equal size, also called folds. The model is trained on all but one fold. The fold not used for training is the validation set used for evaluating the model's performance. This step is

repeated k times until all folds are used exactly once as the validation set. The performance metric is measured at each training round and averaged to compute the final metric value. The benefit of k-fold cross-validation is that the performance metric is more reliable since it stabilizes the variance of accuracy estimates stemming from differences in the training and testing data set [WY20]. When dealing with real-world data sets, tenfold cross-validation is the recommended standard, as it strikes a balance between computational efficiency and reliable performance metrics [Bernd]

2.3 Artificial neural networks

Artificial neural networks have taken the field of machine learning by storm, outperforming previous machine learning models on various tasks, but in particular on natural language processing and computer vision [ON15]. A great advantage of artificial neural networks is the possibility of processing image data in its raw form, whereas more traditional machine learning techniques require features to be extracted from the raw data before it can serve as input for a model. This is especially useful when dealing with image data since it is often hard to extract standardized features from images [LBH15].
While there is a wide variety in the design architecture of artificial neural networks, they all have the same basic structure inspired by the biological brains of humans and other animals. Similar to their biological counterparts, they consist of a large number of interconnected nodes that take an input X, perform an operation F on it, and produce an output Y (similar to how a neuron works)[IGMA18]. An ANN consists of an input layer and an output layer, with hidden layers in between the two.

2.4 ANN depth

The number of neurons and layers, and how the neurons are interconnected are determinative of the character of the network and the performance of the network on different tasks [ON15]. In ANNs, low-order features are learned in shallow layers, and more high-order abstract features are learned in deeper layers within the network. This makes deeper ANNs more suited for complex data containing hierarchical features. Shallow nets on the other hand are a computationally inexpensive and generally better choice when dealing with less complex data [LBH15].

2.5 Convolutional neural networks

Convolutional neural networks are a type of artificial neural network which, as the name prevails, contain convolutional layers. The convolutional layer uses multiple learnable filters called kernels that are usually small in size and always smaller than the input. For each kernel, a unique feature is extracted from the input image, for example, edges or texture. Each kernel slides over the input image horizontally and vertically, calculating the scalar product for each value in the kernel. This is the convolutional operation. Each convolutional operation will result in a unique feature map, a two-dimensional array representative of the strength of the specific feature in the input as detected by the filter [IGMA18].
Apart from Convolutional layers, CNNs contain pooling layers and flattening layers for spatial dimension reduction [IGMA18] [Shy21]. Lastly, Dense layers are characteristic of CNNs, which contain fully connected layers and serve the purpose of finding relationships between different

aggregated features. [DGY+19]
In 3D CNNs, designed for volumetric data and video data, the kernels in convolutional layers are also in 3D shape. They slide over the input data across all three axes (x,y,z). The stride with which they slide over the data can be adjusted independently from one another [SMB19]

2.6 Learning by CNNs

Learning by CNNs starts with forward propagation, where the training data is passed through the network. This results in a prediction of what class the instances belong to. Since the training data is labeled, the accuracy of the model can be assessed using a loss function, which measures the distance between the predictions made by the model and the actual target label. (Li et al., 2014) In this paper, the loss function used is the categorical cross-entropy loss function, which is well suited for multi-class classification tasks [ZS18].
The next step in the learning process is adjusting the parameter so that the loss as calculated by the loss function is minimized. To achieve this, the gradient of the loss function is calculated with the back-propagation algorithm for each node. The optimizer then adjusts the weights and biases in the model intending to minimize the loss function [Bacnd]. The optimizer used in this paper is the Adam optimizer from the torch library.

2.7 Resnet

The ResNet architecture, short for residual network, is a deep CNN characterized by the residual layers in the network. Residual connections skip over a convolutional layer and are directly connected to the next layer [XFZ23]. These connections help mitigate the vanishing gradient problem common in deep neural networks [BK23]. Resnet architecture-based networks have been a staple in the field of computer vision, especially in tasks concerning the classification of medical and biological images [XFZ23].

2.8 MobilenetV2

MobileNetV2 offers a more lightweight solution to classification tasks than ResNet Models. The model contains depth-wise convolutional layers and linear bottleneck layers, which contain significantly fewer nodes than the predeceasing layer. This reduces the number of parameters needed for training, making the model more lightweight. Another characteristic of MobileNetV2 are its inverted residual layers, which first expand the input tensor to a multiple of nodes, then perform convolutional operations on it and finally bring it back to a lower dimension [SHZ+19].

2.9 Swintransformer

The last model discussed in this paper is not a convolutional neural network but a vision transformer model. It works by the concept of shifted windows, meaning the image is processed by dividing it into non-overlapping windows, which are fed into the network and then deeper in the layers, are merged again together for the network to process the global information in the image [HDLG22].

3 Methodology

3.1 Flowchart overview

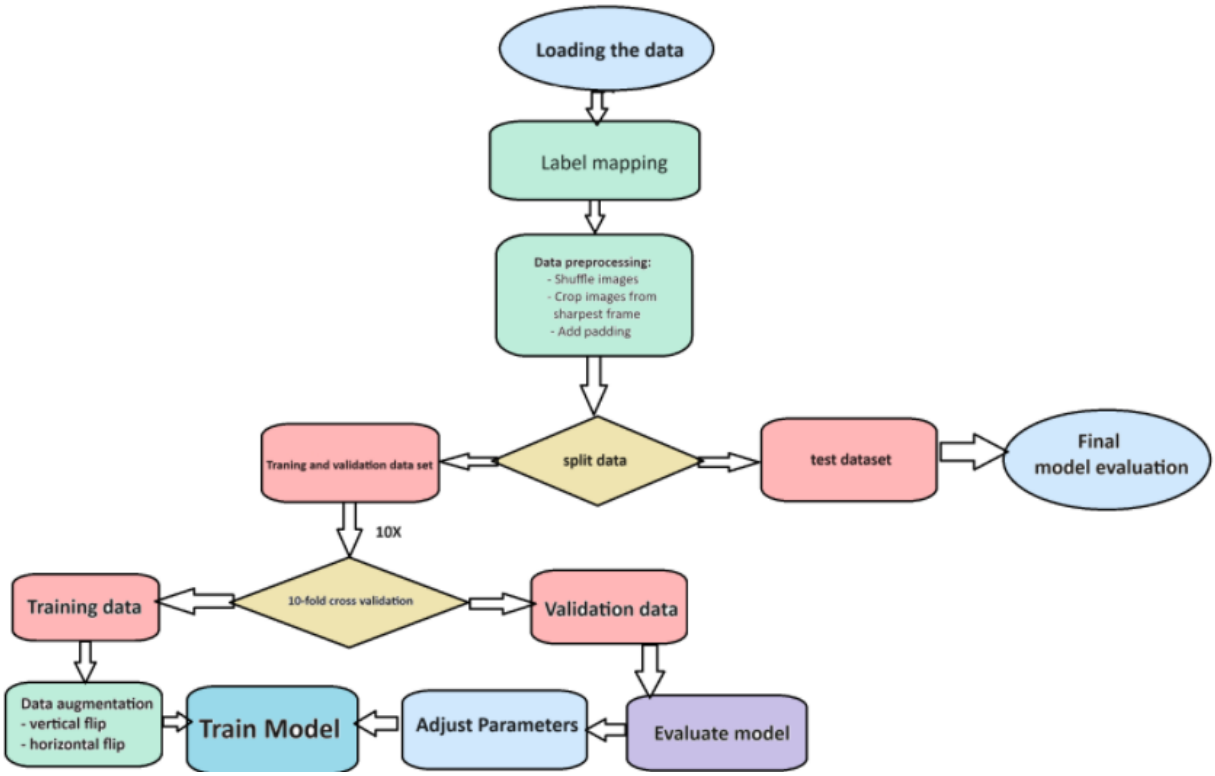

Figure 1: An overview of the methodology of this paper in a flowchart

The elements of the methodology are described in the following paragraphs:

1. Loading the data 3.5
2. label mapping 3.6
3. Data preprocessing: Shuffle images 3.5
4. Data preprocessing: Crop images from the sharpest frame 3.7
5. Data preprocessing: add padding 3.8
6. Split data 3.9
7. 10-fold cross-validation 3.9
8. Data Augmentation: vertical flip, horizontal flip 3.10
9. train model 3.11
10. adjust parameters 3.11
11. final model evaluation 3.12

3.2 Hardware used

Since the training and testing of the model on the data is too computationally demanding for the laptop available during this research (HP Z-book G7 mobile workstation, processor: Intel(R)

Core(TM) i7-10750H CPU @ 2.60GHz, 2592 MHz, 6 core('s), 12 logistic processor(s)), the supercomputing facilities provided by the Universiteit Leiden were used. The ALICE high-performing computer cluster is available for students and employees of both the medical and science faculties and can be accessed through an SSH port. Both CPU and GPU nodes are made available. For this research, the so-called GPU-long nodes were used, consisting of two NVIDIA A100 tensor core GPUs. Each one is good for 40GB of memory.[ALInd].

3.3 Software libaries

Use**Software library name**	**Version**	**Use**
tiffile	2024.2.12	Opening and reading tiff-files
Pickle-mixin	1.0.2	Making objects suitable for conversion to pickle file
Pickleshare	0.7.5	Making, reading, and writing to pickle files
Scikit-learn	1.4.2	Open-source machine learning library for python
Imutils	0.5.4	Provides convenience functions for image processing
Numpy	1.24.3	Package for scientific computing. Provides support for a large collection of mathematical functions
Matplotlib	3.8.0	Library for data visualization
Tdmq	4.64.1	Library providing message queue system
Torch	2.3.1	Open source machine learning library, primarily focused on tasks of computer vision and natural language processing
Torchvision	1.5.1	Library for image transformation
Torchsummary	1.5.1	Library for providing concise summaries of pytorch models
Opency-python	4.9.0.80	Library for computer vision functions. Used in this paper for finding the sharpest layer in the image stack.

Table 1: Software libraries used in this paper

3.4 Data

This dataset is the same as the data used in the paper of Li et al [LPC+23], but then with the difference that the data is kept in its 3D form, namely as a stack of images taken of the pollen from different focal planes. The data is not projected into 2D form, like in the paper of Li et al., where the 3D images of the dataset were compressed by Z-stack projection, which utilizes the standard deviation, the minimum intensity, and extended Focus for each pixel, which were treated like three different channels when fed into the models. While the 3D data contains more information about the depicted pollen grain (every 1.8 μm of the grain is captured, and of course, the sequential order of the stacks also contains information), it does make training the model more computationally expensive [LZW14].

3.5 Preprocessing Pipeline

After the data was loaded from the directory, it was shuffled with a set seed (21). This is done to ensure reproducibility, since it leads to the randomization being consistent across different runs. Subsequently, the data was pre-processed as described in this paragraph. This was done to make the data compatible with the model and to stimulate the learning by the models.

3.6 Label mapping

Since the pollen grains of Parietaria Judaica and Parietaria Officinalis are treated as one class, and the same is true for Urtica Urens and Urtica membranacea the labels of the instances belonging to these classes need to be concatenated into two respective classes. Next, the now three classes are converted to a numerical format (0,1,2)

3.7 Sharpest frame selection and frame cropping

Next, the sharpest frame is selected from the Z-stack using the cv canny filter. The six frames surrounding this sharpest frame serve as the input of the models. The reasoning behind this method is that the sharpest frames will contain the most information about the pollen grain. Including the other frames in the input will waste computational resources and may even add noise, decreasing the model's performance. The sharpest subset is found with the use of the so-called CannyEdgeDetection of the OpenCV library, a popular edge detection algorithm. It is a multistage algorithm, which starts with smoothing the image by sliding a 5x5 Gaussian kernel filter over the image. This is done to reduce the noise in the image. Next, a Sobel filter is put over the image, in horizontal (Gx) and vertical G(y) directions. The magnitude and direction/angle are then computed as follows:

$$\text{Edge\_Gradient}(G) = \sqrt{G_x^2 + G_y^2} \tag{1}$$

$$\text{Angle}(\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right) \tag{2}$$

The next step is non-maximum suppression, which entails checking for each pixel whether the pixel is a minimum or a maximum in its gradient direction. When this is not the case, the pixel is set to zero. With double thresholding, edges are identified as strong (above the high threshold) or weak (between the low and high threshold). Pixels that fall below the strong threshold are considered non-edges. In this paper, a value of 100 is used as the low threshold, and 300 as the strong.
The final step is Hysteresis, where only the weak edges connected to strong edges are retained, while others are discarded [Kal15]
The frame with the most edges is considered the sharpest frame. This is the frame around which the other frames are taken. The six surrounding frames are then selected, if not out of the images' range. If the sharpest frame is too close to the perimeters of the image, the range is shifted to the opposing side of the image, to keep the number of frames at six.

3.8 Padding and cropping

Since the models only take input images of size 224 x 224 x 6 (height, width, depth)and the images in the dataset differ in size, they need to be cropped or extended to these target dimensions. Images that are too small are provided with padding. Pixels are added to the image and assigned the mean grayscale value of that image. The choice for the mean value instead of zero padding stems from the desire to add as little extra information to the image as possible. This goal is also the reason for using padding in general and not simply zooming in on the smaller images. The size of the pollen grain information that can be used by the models during the classification.
Images whose dimensions exceed those expected by the models are cropped. The target height and width are deliberately chosen to be greater than those of the majority of the original images of the dataset to prevent loss of information from cropping.

3.9 Splitting into folds

The next step in the preprocessing of the data is to split the data into a training, validation, and test set. The test set is the first to be separated from the training and validation set. 10 percent of the data is kept aside for this purpose. The custom class 'dataloaders' contains the code where the data division is made. At every fold, a new training and validation split is made with a set seed, to prevent data leakage and for reproducibility 3.5. The training data is then processed further with data augmentation while the validation set is not. The train/validation ratio is 9 to 1. The split is performed with the help of Pytorch's Datasets class. These datasets are then wrapped into instances of the class Dataloader that will further handle the data augmentation as described in the paragraph below 3.10.

3.10 Data augmentation

Firstly, all gray-scale values in the image are normalized between $(-1, 1)$. This is done to prevent single, outlier features from becoming too dominant in the image. Additionally, deep neural networks tend to show better performance and accelerated convergence when input scales are normalized.[PTK+20]. Normalizing to $(-1, 1)$ is particularly beneficial because it centers the data around zero, promoting a more stable gradient flow and faster convergence during training when using a ReLu activation function.The hyper-parameter 'augthreshold' is decisive in the amount of data augmentation performed on the training data set. The threshold is set to a number between 0 and 1. For each image in the training set, a random number between 0 and 1 is generated. If the new number is higher than the threshold, a copy of the image is used for data augmentation. A high augmentation threshold will lead to little data augmentation, and vice versa.

The data augmentation is performed by vertically and horizontally flipping the images to increase the diversity of the training dataset. The main idea behind this is to improve the generalization process of the model and reduce over-fitting. Data augmentation is particularly valuable when dealing with little training data since new data points are artificially created [Artnd]. The reasoning behind only performing horizontal and vertical flips was that other operations like rotation might add non-existent information to the images or result in a loss of information, like the pixels needed to fill in the corners of the image after rotation.

3.11 Training and Validation

Once an augmented training set and a validation set are acquired the model is trained. The function 'getmodel' initializes the model architecture for each fold. The model is then transferred to the GPU to reduce training time.

The training loop consists of 30 epochs for each fold. The training data is fed into the model in batches, which aids efficiency and structured learning. The learning by the model takes place as described in paragraph 2.6 (forward pass, loss computation, backward pass, and parameter update per epoch). After the parameter update, the model generated by the epochs training is evaluated on the validation set. The 'evaluate()' method calculates the accuracy, loss, and F1 score of the model on the training and validation set. If the models' accuracy on the validation set is the highest of the current fold yet, the models' parameters are saved into a '. pth' file using the 'savecheckpoint' method. Additionally, the training and validation metrics are saved to a pickle file.

The training loop is finished when for each fold, a model, trained and validated on all epochs is saved.

3.12 Testing

Once the model with the highest validation accuracy is saved for each fold, the testing can begin. In the 'datalaoders' class a distinct division was made between data for training and validation and the final test data. It is important for the test data to not be seen before by the model, to achieve a reliable accuracy score. This is ensured by the 'testmodel' function which will return an error message if instances from the test data set are found in the training or validation data. When data leakage is ruled out, the models are tested on the unseen test data. The accuracy on the test set is taken for all the folds models and averaged, resulting in a robust and reliable metric.

3.13 Plotting

Finally, the results are plotted. The average train and validation loss are computed for each epoch overall folds, after which they are plotted in a graph. The graph is saved. The average and best accuracy on the test data of the models overall folds is added to the graphs' titles.

3.14 Code Adaptation

Since this thesis is the continuation of a previous student's work, a significant amount of the code used for this thesis was re-used from her prvious work. This is mainly the case for the code containing the preprocessing and data augmentation of the image data. This student's work can be found in the bibliography at [Seb23].
Additionally, there was a collaboration with fellow student Tijs Konijn, also mainly on the code concerning the data pre-processing.

Lastly, the building of the MobileNetv2 model was done with another author's implementation of a 3D version of MobileNetV2 as a guideline. The implementation authors' model was made to

exactly match the original 2d models' architecture as described in the developer's paper [HZC+17], except in 3D.
The implementation of this model can be found in an open-source project available on GitHub kopuklu2019mobilenet. The original code was developed by Okan Köpüklü and can be found at his public GitHub Repository. The code used for this project has a different structure of classes and functions as well as different naming but shows similarities in layer configuration since both codes eventually result in the same model, namely a 3D version of [HZC+17].

4 Results

In the section below, the results for the three models are discussed. By adjusting the hyper-parameter settings, the goal was to maximize the accuracy of the models on the test dataset.

4.1 Baseline ResNet3D

The baseline model used in this paper is ResNet 3D. Optimal parameter settings were obtained from previous work [Seb23], with an exception for the number of frames, which was set to 6. The settings are the following:

hyper-parameter	value
folds	10
epochs	30
learning rate	0.0001
augmentation threshold	0.5
number of frame	6
batch size training	16
batch size validation	16

Table 2: hyper-parameter settings on the ResNet 3D model

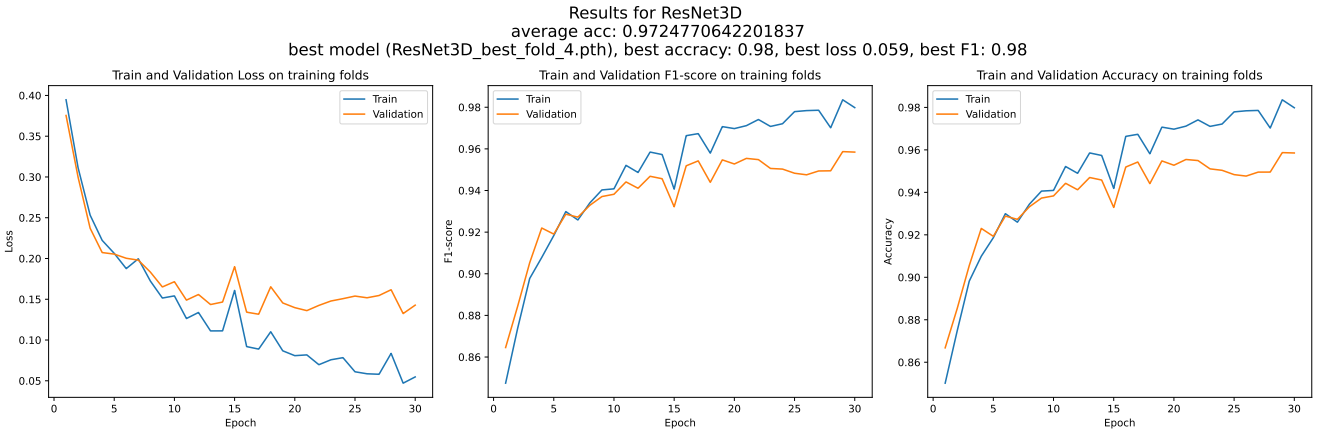

Figure 2: Average training and validation loss, F1-score and accuracy over all folds per epoch for the ResNet3D model trained from scratch with hyper-parameter settings 2

metric	Accuracy	F1-score	Loss
average value	0.972	0.972	0.074

Table 3: average performance metrics on the test set for the ResNet3D model trained from scratch with hyper-parameter settings 2

pre-trained ResNet 3D Using the same hyper-parameter settings as 2, the pre-trained ResNet 3D yielded the following results:

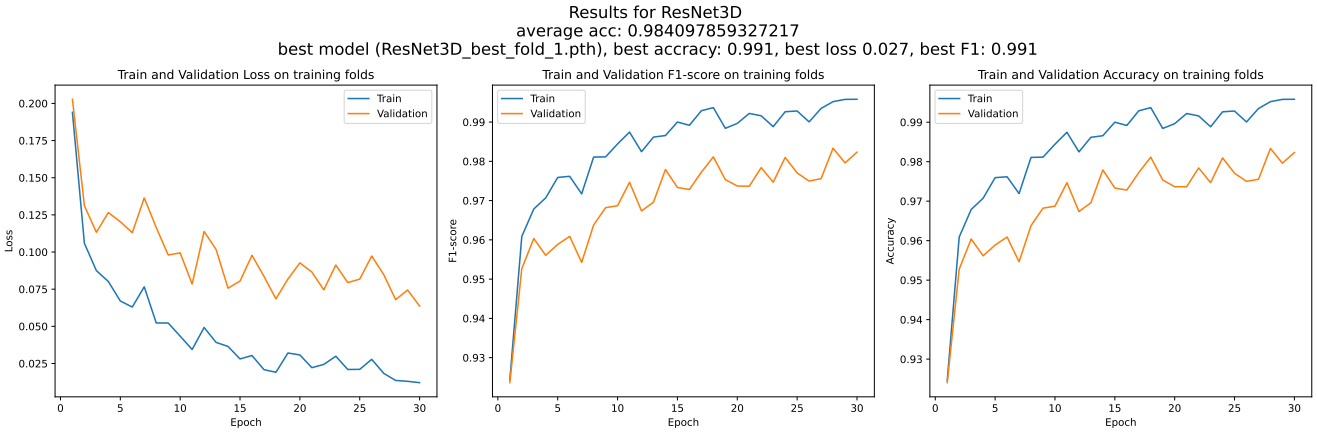

Figure 3: Average training and validation loss, F1-score and accuracy over all folds per epoch for the pre-trained ResNet3D model with hyper-parameter settings 2

metric	Accuracy	F1-score	Loss
average value	0.984	0.984	0.067

Table 4: average performance metrics on the test set for the pre-trained ResNet3D model with hyper-parameter settings 2

4.2 Swintranformer 3D

For the specific type of Swintranformer researched in this paper, no pre-trained weights were found online. Therefore, only the Swintransformer 3D trained from scratch is discussed.

First run The starting point for finding the optimal settings for the 3D Swintransformer model were those found to be optimal for the ResNet3D model. The hyper-parameters for the number of attention heads and window size were set to default.

hyper-parameter	value
folds	10
epochs	30
learning rate	0.0001
augmentation threshold	0.5
number of frame	6
batch size training	16
batch size validation	16
number of heads	[3, 6, 12, 24]
Window size	[8,7,7]

Table 5: hyper parameter settings on the Swintranformer 3D models first run

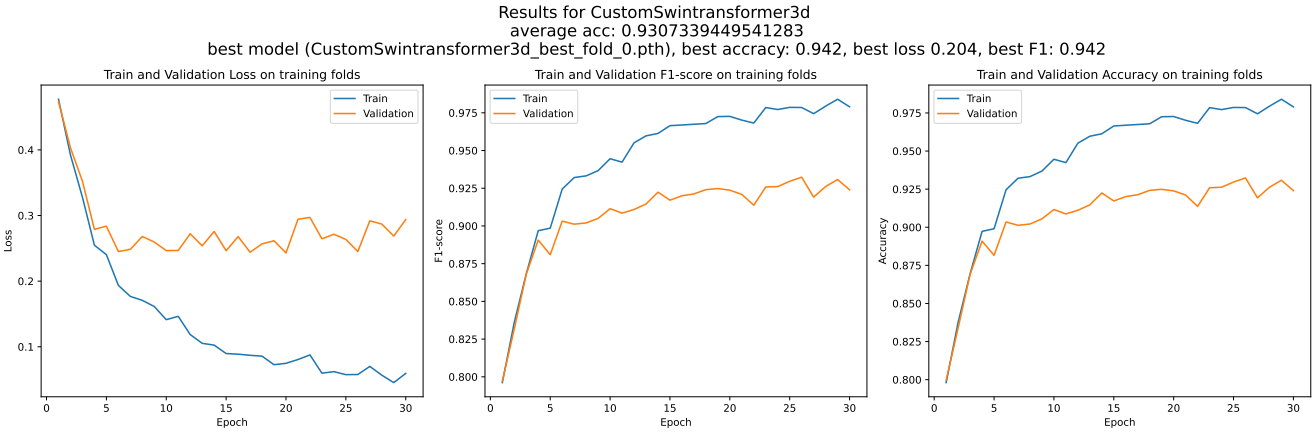

Figure 4: Average training and validation loss, F1-score and accuracy over all folds per epoch for the Swintransformer 3D model trained from scratch

metric	Accuracy	F1-score	Loss
average value	0.931	0.924	0.224

Table 6: average performance metrics on the test set for the Swintranformer 3D model trained from scratch

Augmentation threshold The effect of adjusting the augmentation threshold was examined, using the following hyper parameter settings:

hyper-parameter	value
folds	10
epochs	30
learning rate	0.0001
augmentation threshold	**[0.1, 0.2, 0.5, 0.7]**
number of frames	6
batch size training	16
batch size validation	16
number of heads	[3, 6, 12, 24]
Window size	[8,7,7]

Table 7: hyper parameter settings on the Swintranformer 3D model for adjusting the augmentation threshold

augmentation threshold	0.1	0.2	0.5	0.7
Mean accuracy	0.931	0.823	0.931	0.919

Table 8: Mean accuracy scores for different augmentation threshold values

The best setting for the augmentation threshold is 0.5. The performance of the model on settings 7 and augmentation threshold 0.5 is depicted in 4 The worst setting found for the augmentation threshold was 0.7. The performance graph for these settings can be found in Appendix 1 8:

Training batch size The effect of adjusting the training batch size was examined, using the following hyper-parameter settings:

hyper-parameter	value
folds	10
epochs	30
learning rate	0.0001
augmentation threshold	0.2
number of frames	6
batch size training	**[16, 20, 50, 70]**
batch size validation	16
number of heads	[3, 6, 12, 24]
Window size	[8,7,7]

Table 9: hyper-parameter settings on the Swintranformer 3D model for adjusting the training batch size

yielding the following results:

Training batch size	16	20	50	70
Mean accurracy	0.823	0.930	0.933	0.931

Table 10: Mean accuracies for different training batch sizes

The best setting for training batch size is 50. The performance graph of this setting can be found in Appendix 9
The worst setting found was 16. The performance graph for these settings can be found in appendix 10

Validation batch size The effect of adjusting the validation batch size was examined, using the following hyper parameter settings: yielding the following results:

hyper-parameter	value
folds	10
epochs	30
learning rate	0.0001
augmentation threshold	0.2
number of frames	6
batch size training	20
batch size validation	**[16, 20, 50, 70]**
number of heads	[3, 6, 12, 24]
Window size	[8,7,7]

Table 11: hyper parameter settings on the Swintranformer 3D model for adjusting the validation batch size

validation batch size	16	20	50	70
Mean accurracy	0.930	0.935	0.933	0.9325

Table 12: Mean accuracies for different validation batch sizes

The best setting for validation batch size is 20. The performance graph for these settings can be found in Appendix 11 The worst setting found was 16. The performance graph for these settings can be found in Appendix 12

Learning rate The effect of adjusting the learning rate was examined, using the following hyper-parameters:

hyper-parameter	value
folds	10
epochs	30
learning rate	**[0.001, 0.0001, 0.00001]**
augmentation threshold	0.2
number of frame	6
batch size training	16
batch size validation	16
number of heads	[3, 6, 12, 24]
Window size	[8,7,7]

Table 13: hyper parameter settings on the Swintranformer 3D model for adjusting the learning rate

Learning rate	0.001	0.0001	0.00001
Mean accurracy	0.431	0.823	0.653

Table 14: The effect of adjusting the learning rate on the Swintransformer 3D model

The best setting for the learning rate is 0.0001. For a graph of the best settings see Appendix 10. The worst setting found was 0.001. The performance graph of the worst settings is depicted in Appendix 13:

Optimized parameters from the paragraphs above, the following optimal hyper-parameter settings arise:

hyper-parameter	value
folds	10
epochs	30
learning rate	0.0001
augmentation threshold	0.5
number of frame	6
batch size training	50
batch size validation	20
number of heads	[3, 6, 12, 24]
Window size	[8,7,7]

Table 15: seemingly optimal hyper parameter settings on the Swintranformer 3D model

When running the model on these settings, the following results were obtained:

metric	Accuracy	F1-score	Loss
average value	0.9243	0.9241	0.2860

Table 16: average performance metrics on the test set for the MobileNetv2 model trained from scratch with parameter settings 15

The performance graphscan be found in Appendix 14

4.3 MobileNetV2 3D

First run The starting point for finding the optimal settings for the 3D MobilenetV2 model were those found to be optimal for the ResNet3D model. The following hyper-parameter settings were used:

hyper-parameter	value
folds	10
epochs	30
learning rate	0.0001
augmentation threshold	0.5
number of frame	6
batch size training	16
batch size validation	16

Table 17: hyper parameter settings on the MobileNetv2 3D models' first run

This resulted in the following performance metrics:

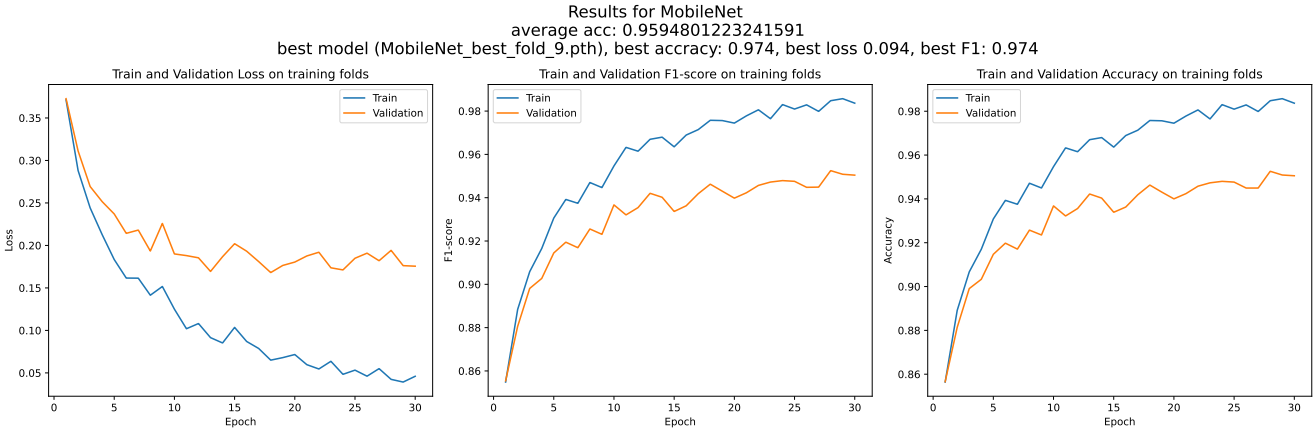

Figure 5: Average training and validation loss, F1-score and accuracy over all folds per epoch for the MobileNetv2 3D model trained from scratch with parameter settings 17

metric	Accuracy	F1-score	Loss
average value	0.959	0.959	0.1308

Table 18: average performance metrics on the test set for the MobileNetv2 model trained from scratch with parameter settings 17

Augmentation threshold The effect of adjusting the augmentation threshold was examined, using the following hyper parameter settings:

hyper-parameter	value
folds	10
epochs	30
learning rate	0.0001
augmentation threshold	**[0.1, 0.2, 0.5, 0.7]**
number of frames	6
batch size training	16
batch size validation	16

Table 19: hyper parameter settings on the MobileNetv2 3D model for adjusting the augmentation threshold

augmentation threshold	0.1	0.2	0.5	0.7
Mean accuracy	0.936	0.937	0.959	0.919

Table 20: Mean accuracy scores of MobileNetv2 3D for different augmentation threshold values using parameter settings 19

The best setting for the augmentation threshold is 0.5. The performance graph for these settings is depicted in 5. The worst setting found was 0.7. The performance graph of the worst settings is depicted in Appendix 16

Training batch size The effect of adjusting the training batch size was examined, using the following hyper-parameter settings:

hyper-parameter	value
folds	10
epochs	30
learning rate	0.0001
augmentation threshold	0.5
number of frames	6
batch size training	**[16, 20, 50, 70]**
batch size validation	16

Table 21: hyper parameter settings on the MobileNetv2 3D model for adjusting the training batch size

training batch size	16	20	50	70
Mean accuracy	0.959	0.919	0.903	0.899

Table 22: Mean accuracy scores for different training batch sizes

The best setting for the training batch size is 16. For a graph of the best settings see 5. The worst setting found was 70. The performance graph of the worst settings is depicted in Appendix 17

validation batch size The effect of adjusting the validation batch size was examined, using the following hyper-parameter settings:

hyper-parameter	value
folds	10
epochs	30
learning rate	0.0001
augmentation threshold	0.5
number of frames	6
batch size training	16
batch size validation	**[16, 20, 50, 70]**

Table 23: hyper parameter settings on the MobileNetv2 3D model for adjusting the validation batch size

validation batch size	16	20	50	70
Mean accuracy	0.959	0.922	0.925	0.924

Table 24: Mean accuracy scores for different validation batch sizes

The best setting for the validation batch size is 16. For a graph of the best settings see 5. The worst setting found was 20. The performance graph of the worst settings is depicted in Appendix 18:

Learning rate The effect of adjusting the learning rate was examined, using the following hyper-parameter settings:

hyper-parameter	value
folds	10
epochs	30
learning rate	**[0.01, 0.001, 0.0001, 0.00001]**
augmentation threshold	0.5
number of frames	6
batch size training	16
batch size validation	16

Table 25: hyper parameter settings on the MobileNetv2 3D model for adjusting the learning rate

Learning rate	0.01	0.001	0.0001	0.00001	
Mean accuracy	0.922	0.959	0.959	0.413	

Table 26: Mean accuracy scores for different learning rate values

The best setting for the learning rate is 0.0001. For a graph of the best settings see 5. The worst setting found was 0.00001, the performance graph is depicted in Appendix 19

pre-trained Mobilenetv2 3D The performance of the pre-trained mobilenet on hyper-parameter settings 17, yielded the following results:

metric	Accuracy	F1-score	Loss
average value	0.950	0.950	0.146

Table 27: average performance metrics on the test set for the pre-trained MobileNetv2 model with parameter settings 17

The performance graph can be found in Appendix 20

Optimal settings for pre-trained model The best parameter settings found for the pre-trained model were the following :

hyper-parameter	value
folds	10
epochs	30
learning rate	0.001
augmentation threshold	0.2
number of frames	6
batch size training	16
batch size validation	20

Table 28: best hyper-parameter settings found for the pre-trained MobileNetv2 3D model

These settings yielded the following results:

metric	Accuracy	F1-score	Loss
average value	0.974	0.974	0.073

Table 29: average performance metrics on the test set for the pre-trained MobileNetv2 model with parameter settings 28

The performance graph can be found in Appendix 21

4.4 Results overview

For each model, trained and pre-trained the parameters which resulted in the highest average accuracy score on the test set were used again, but this time on 50 epochs instead of 30. The accuracy scores are depicted in par 30 . The performance graphs are depicted the Appendix at:

1. ResNet 3D 6
2. ResNet 3D (pre-trained) 7
3. Swintransformer 15
4. MobileNetV2 22

5. MobileNetV2 (pre-trained) 23

Listed below are the best-acquired results for each model on 30 epochs and 50 epochs, trained from scratch and pre-trained. The accuracy is the accuracy the model yielded on the test set. The best-performing model is the pre-trained ResNet model on 50 epochs, which scores an accuracy of 0.989.

Model	**Best Avg Accuracy 30 e**	**Best Avg Accuracy 50 e**	**Hyper-param**
Resnet 3D	0.972	0.988	2
Resnet 3D (pre-trained)	0.984	0.989	2
Swintransformer 3D	0.935	0.931	5
MobileNetv2 3D	0.959	0.936	5
MobileNetv2 3D (pre-trained)	0.974	0.956	28

Table 30: Best performances of all models on 30 epochs and 50 epochs (pre-trained and from scratch)

5 Discussion and Conclusion

Although the papers by Huang [HDLG22] and [ZCL+24] raised high expectations for the Swin Transformer model on the task of pollen classification with sequential data, its performance was disappointing. Part of the reason for this is, of course, the lack of pre-trained weights available for this model. However, when looking at table 30, it becomes clear that the Swin Transformer model generally performs worse than ResNet and MobileNet, even when trained from scratch.
Swintranformer benefits from slightly larger batch sizes than ResNet, possibly because the Transformer models generally respond well to the stabilizing effects of larger batch sizes. Its architecture, with complex attention mechanisms and hierarchical processing units, makes it more sensitive to unstable gradients. [PB18]. The Swintrainsformer did not benefit from longer training, suggesting over-fitting when trained on 50 epochs.
The MobileNetv2 model performs reasonably well, considering its light weight and few parameters. Using a mobileNetv2 for pollen classification could be desirable if reducing computational cost is one's special interest. The pre-trained MobileNet outperforms the model trained from scratch by 1.5 percent points. Interestingly, the hyper-parameter settings working the best for the model trained from scratch are not those resulting in the best results for the pre-trained model. Seemingly, the fact that the pre-trained model benefits from a higher learning rate is the effect of it already having already captured general features from a large dataset. The model will not suffer from a destabilized learning process, which is often a result of a high learning rate. The pre-trained model is already close to a good solution so faster adjustments are beneficial.[ST17]. The MobileNet did not benefit from longer training, suggesting the model over-fits on the training data when trained for 50 epochs.

When looking at the performance graphs of the pre-trained ResNet, it becomes clear that the pre-trained models are less susceptible to over-fitting, since the training and validation accuracy over the epochs 4 lie closer together than those of the models trained from scratch 2. A high accuracy score on the training set and a low accuracy on the validation set implies over-fitting.

This is also the case for the MoblieNetV2 models 5 20. The ResNet model is the only model which did benefit from longer training, improving the accuracy scores with 0.016 and 0.004. Since these differences are only small, one could wonder if it is worth spending the extra computational power on this exra learning. The answer to this question will depend on one's preferences and available resources.

5.1 Conclusion

The highest accuracy achieved in this paper was by the pre-trained ResNet model, trained on 50 epochs, achieving an accuracy of 0.989. This is not as high as the highest scores discussed in the Literature Review 1.2, but it ranks around the average of all the scores. This is a promising sign for pollen classification with 3D data.
Depending on the available computational resources, the best models found in this paper to use for pollen classification are the pre-trained ResNet model (large computational demand) or the MobileNetV2 (small computational demand).

5.2 Future work

Since the field of pollen classification is broad and diverse, the research conducted in this paper could be extended to achieve better performance metrics. Some suggestions for this goal to be achieved are mentioned in the subsections below.

5.3 Pre-trained Swintransformer

At the time of this research, no pre-trained weights for the specific Swintransformer3d class from PyTorch were publicly available. Since the pre-trained versions of the other two models performed well, it would be worth the effort to try a pre-trained Swintransformer on this task once the weights are made available.

5.4 Experimenting with number of frames

In this paper, a number of six frames is used from each image. It would be interesting to see whether including more or fewer frames in the input results in better or worse performance. On the one hand, the blurry peripheral frames might only add noise and therefore lead to worse performance. On the other hand, the data might contain information that is not visible to the naked (layman's) eye but can be very well extracted by the model, which would lead to an improvement in performance.

5.5 Colored images

Many of the papers discussed in the Literature Review 1.2 use colored images as the model's input. It would be worthwhile investigating whether the extra computational costs associated with this change would outweigh the potential increase in performance.

References

[ALInd] About alice - hpc wiki, n.d. accessed 7.21.24.

[Artnd] The art of data augmentation: Journal of computational and graphical statistics: Vol 10, no 1, n.d. accessed 7.23.24.

[Bacnd] Backpropagation in a neural network: Explained, n.d. accessed 7.18.24.

[Bernd] Berrar_ebcb_2nd_edition_cross-validation_preprint.pdf, n.d.

[BK23] L. Borawar and R. Kaur. Resnet: Solving vanishing gradient in deep networks. In R.P. Mahapatra, S.K. Peddoju, S. Roy, and P. Parwekar, editors, *Proceedings of International Conference on Recent Trends in Computing*, pages 235–247. Springer Nature, Singapore, 2023.

[Boond] [bookchapter14]transfer learning.pdf, n.d.

[BOT+20] S. Battiato, A. Ortis, F. Trenta, L. Ascari, M. Politi, and C. Siniscalco. Detection and classification of pollen grain microscope images. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 4220–4227, 2020.

[CPIM18] G. Ciprandi, P. Puccinelli, C. Incorvaia, and S. Masieri. Parietaria allergy: An intriguing challenge for the allergist. *Medicina*, 54:106, 2018.

[Dao18] Amar Idrees Daood. *Pollen Grains Recognition Based on Computer Vision Methods*. phdthesis, Florida Institute of Technology, Melbourne, Florida, 2018. A dissertation submitted to the College of Engineering in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Engineering, Spring 2018.

[DCNMO+20] G. D'Amato, H.J. Chong-Neto, O.P. Monge Ortega, C. Vitale, I. Ansotegui, N. Rosario, T. Haahtela, C. Galan, R. Pawankar, M. Murrieta-Aguttes, L. Cecchi, C. Bergmann, E. Ridolo, G. Ramon, S. Gonzalez Diaz, M. D'Amato, and I. Annesi-Maesano. The effects of climate change on respiratory allergy and asthma induced by pollen and mold allergens. *Allergy*, 75:2219–2228, 2020.

[DDS+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[DGY+19] J. Dolz, K. Gopinath, J. Yuan, H. Lombaert, C. Desrosiers, and I. Ben Ayed. Hyperdense-net: A hyper-densely connected cnn for multi-modal image segmentation. *IEEE Transactions on Medical Imaging*, 38:1116–1126, 2019.

[HDLG22] X. Huang, M. Dong, J. Li, and X. Guo. A 3-d-swin transformer-based hierarchical contrastive learning method for hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 60:1–15, 2022.

[HZC+17] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017.

[IGMA18] S. Indolia, A.K. Goswami, S.P. Mishra, and P. Asopa. Conceptual understanding of convolutional neural network- a deep learning approach. *Procedia Computer Science*, 132:679–688, 2018.

[Kal15] Pradeep Kalra. Edge detection using the canny algorithm, 2015. Accessed: 2024-07-23.

[LBH15] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436–444, 2015.

[LPC+23] Chao Li, Maarten Polling, Lei Cao, Barbara Gravendeel, and F.J. Verbeek. Analysis of automatic image classification methods for urticaceae pollen classification. *Neurocomputing*, 522:181–193, 2023.

[LZW14] H. Li, R. Zhao, and X. Wang. Highly efficient forward and backward propagation of convolutional neural networks for pixelwise classification. 2014.

[ON15] K. O'Shea and R. Nash. An introduction to convolutional neural networks, 2015.

[PB18] Martin Popel and Ondřej Bojar. Training tips for the transformer model. *The Prague Bulletin of Mathematical Linguistics*, 110:43–70, 2018.

[PLC+21] M. Polling, C. Li, L. Cao, F. Verbeek, L.A. de Weger, J. Belmonte, C. De Linares, J. Willemse, H. de Boer, and B. Gravendeel. Neural networks for increased accuracy of allergenic pollen monitoring. *Scientific Reports*, 11:11357, 2021.

['Po21] 'Polling'. *'The hidden biodiversity of pollen'*. PhD thesis, 'University of Oslo', 2021.

[PTK+20] N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, and A. Iosifidis. Deep adaptive input normalization for time series forecasting. *IEEE Transactions on Neural Networks and Learning Systems*, 31:3760–3765, 2020.

[SCH16] WILLIAM SCHEPPEGRELL. HAY-FEVER: ITS CAUSE AND PREVENTION. *Journal of the American Medical Association*, LXVI(10):707–712, 03 1916.

[Seb23] Shreya Sebastian. Enhancing pollen classification accuracy through the use of whole stack images in 3d convolutional neural networks, 2023. Accessed: 2023-08-23.

[Shy21] R. Shyam. Convolutional neural network and its architectures. 2021.

[SHZ+19] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. 2019.

[SMB19] R.D. Singh, A. Mittal, and R.K. Bhatia. 3d convolutional neural network for object recognition: a review. *Multimedia Tools and Applications*, 78:15951–15995, 2019.

[ST17] Leslie N. Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates. *arXiv preprint arXiv:1708.07120*, 2017. Preprint. Work in progress.

[WY20] Tzu-Tsung Wong and Po-Yang Yeh. Reliable accuracy estimates from k-fold cross validation. *IEEE Transactions on Knowledge and Data Engineering*, 32(8):1586–1594, 2020.

[XFZ23] W. Xu, Y.-L. Fu, and D. Zhu. Resnet and its application to medical image processing: Research progress and challenges. *Computer Methods and Programs in Biomedicine*, 240:107660, 2023.

[ZCL+24] B. Zu, T. Cao, Y. Li, J. Li, F. Ju, and H. Wang. Swint-srnet: Swin transformer with image super-resolution reconstruction network for pollen images classification. *Engineering Applications of Artificial Intelligence*, 133:108041, 2024.

[ZS18] Z. Zhang and M.R. Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. 2018.

6 Appendix

6.1 Performance graph ResNet3D

Figure 6: Performance graph of RESNET 3D model trained from scratch with optimal settings trained on 50 epochs

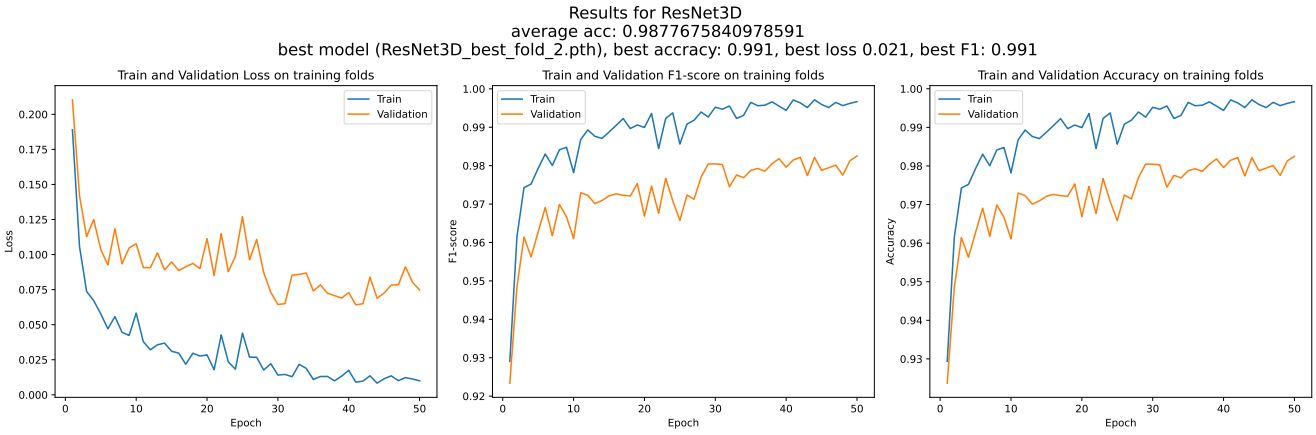

Figure 7: Performance graph of the pre-trained RESNET 3D with optimal settings trained on 50 epochs

6.2 Performance Graphs Swintransformer

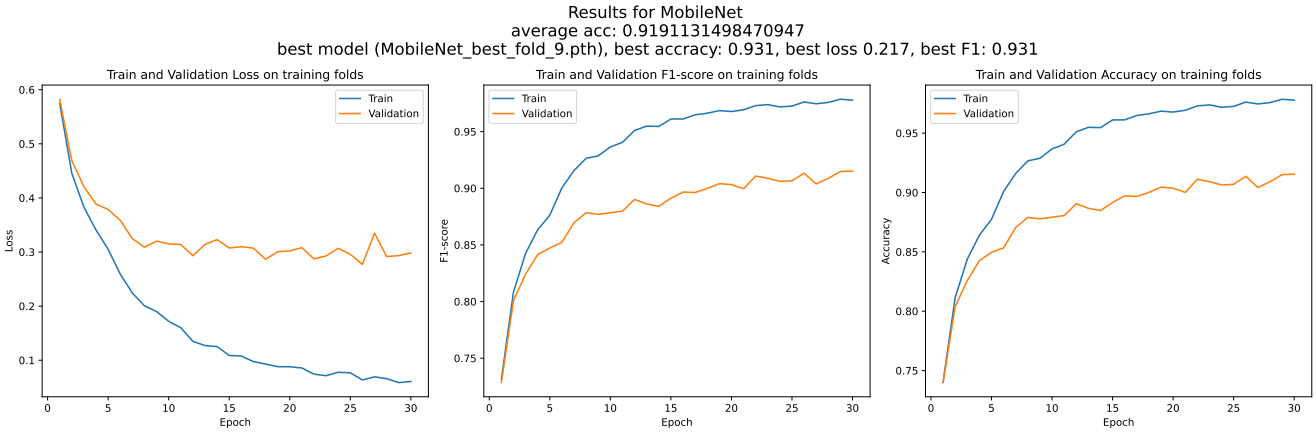

Figure 8: Graphs of the loss, F1 score and accuracy of the Swintransformer 3D model on the worst setting found: 0.7

Figure 9: Graphs of the loss, F1 score and accuracy of the Swintransformer 3D model on the best setting found for training batch size: 50

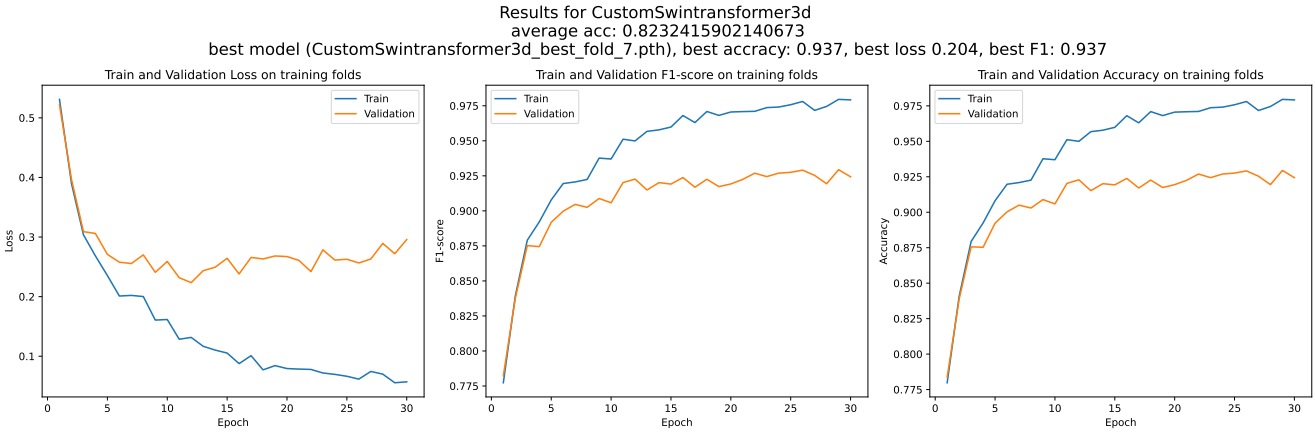

Figure 10: Graphs of the loss, F1 score and accuracy of the Swintransformer 3D model on the worst setting found for training batch size: 16 and the best setting found for learning rate: 0.0001

Figure 11: Graphs of the loss, F1 score, and accuracy of the Swintransformer 3D model on the best setting found for validation batch size: 20

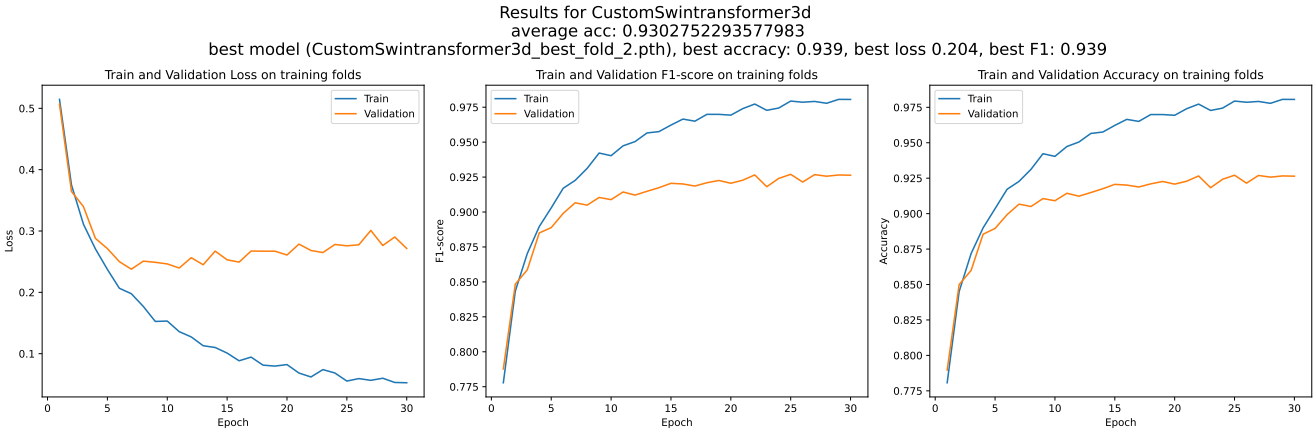

Figure 12: Graphs of the loss, F1 score, and accuracy of the Swintransformer 3D model on the worst setting found for validation batch size: 16

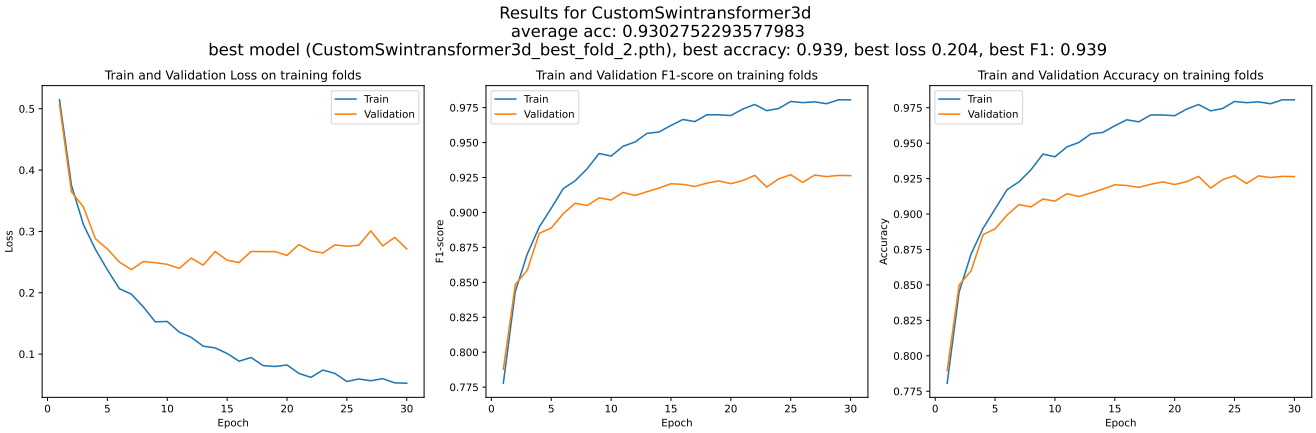

Figure 13: Graphs of the loss, F1 score, and accuracy of the Swintransformer 3D model on the worst setting found for learning rate: 0.001

Figure 14: Swintranformer model ran on hyper-parameter settings 15

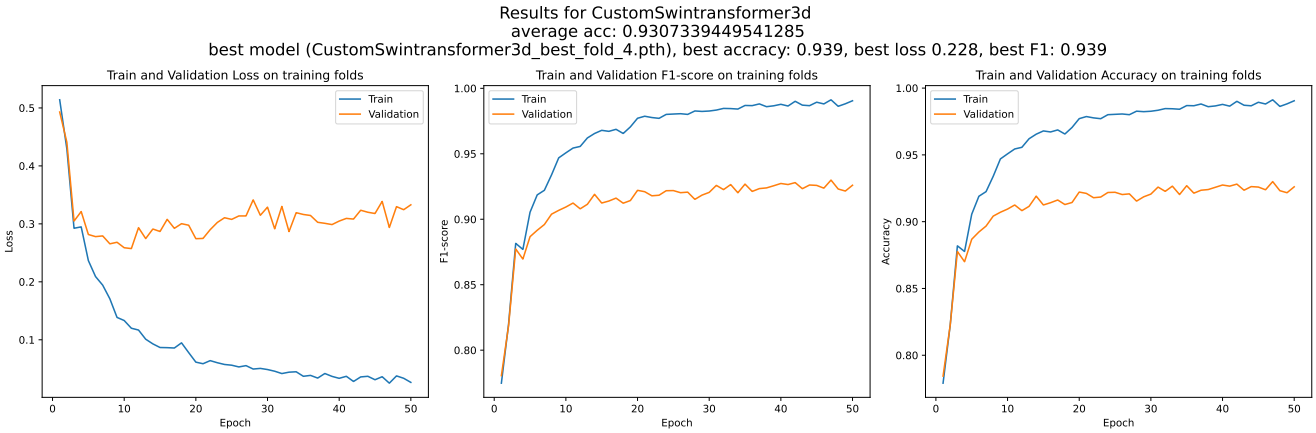

Figure 15: Performance graph of the Swintransformer model trained on 50 epochs with optimal parameter settings

6.3 MobileNetV2 performance graphs

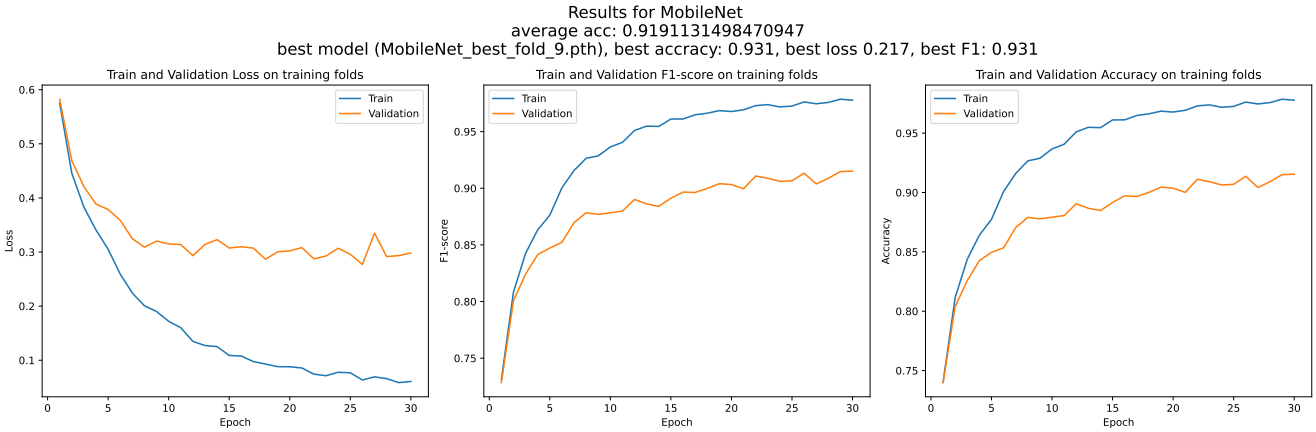

Figure 16: Graphs of the loss, F1 score, and accuracy of the MobileNetV2 3D model on the worst setting found for the augmentation threshold: 0.7

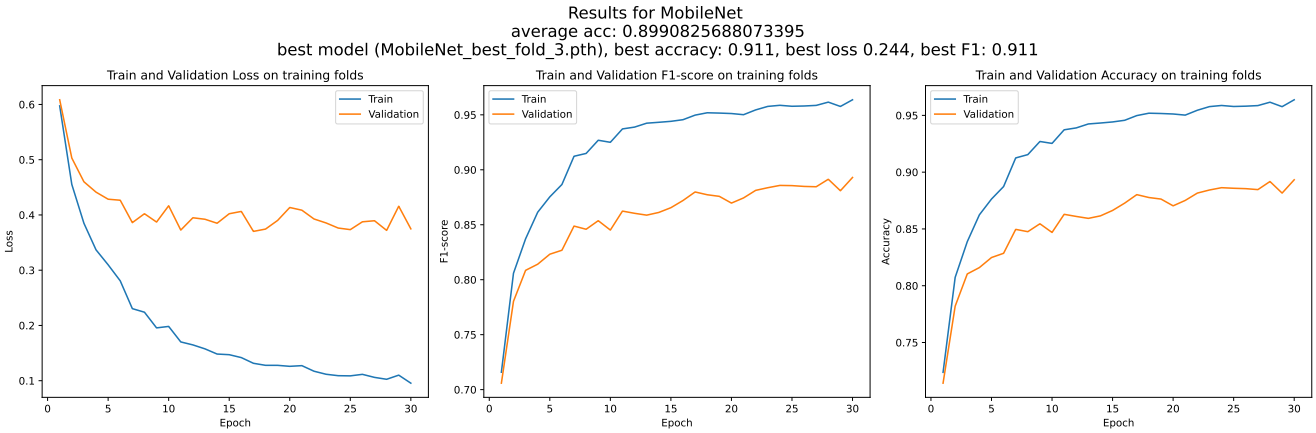

Figure 17: Graphs of the loss, F1 score, and accuracy of the MobileNetV2 3D model on the worst setting found for the training batch size 70

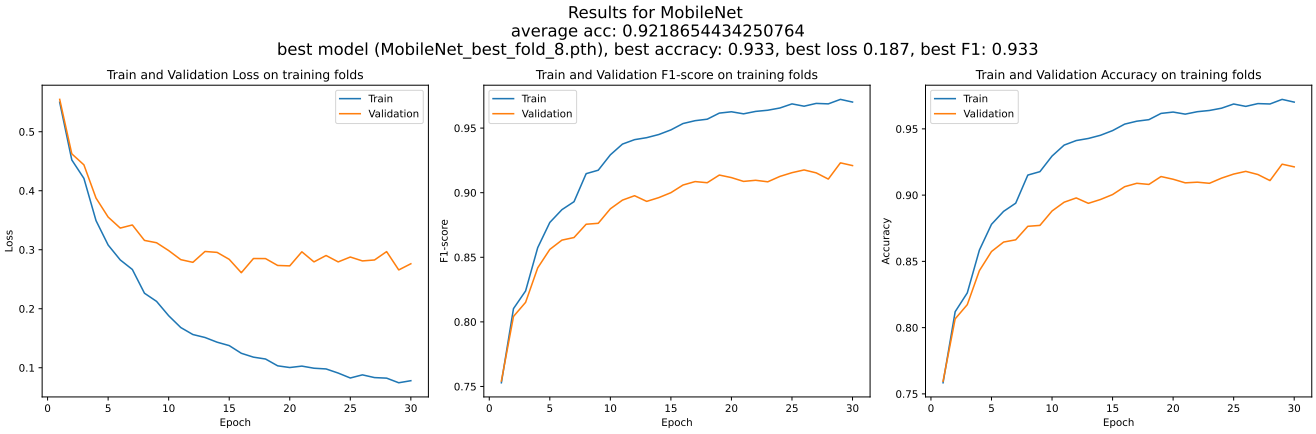

Figure 18: Graphs of the loss, F1 score, and accuracy of the MobileNetV2 3D model on the worst setting found for validation batch size: 20

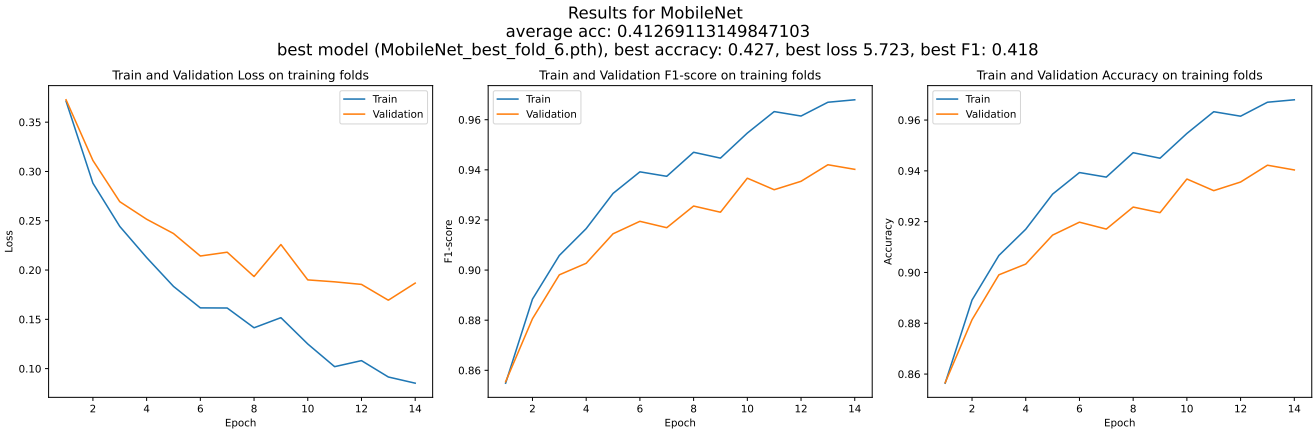

Figure 19: Graphs of the loss, F1 score, and accuracy of the MobileNetV2 3D model on the worst setting found for the learning rate 0.00001

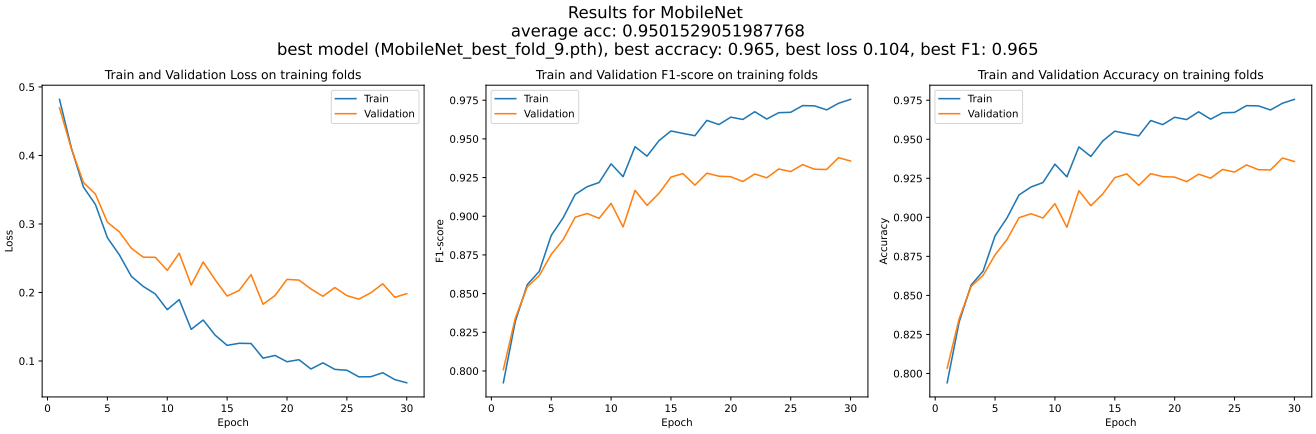

Figure 20: Average training and validation loss, F1-score and accuracy over all folds per epoch for the pre-trained MobileNetv2 3D model parameter setting 5

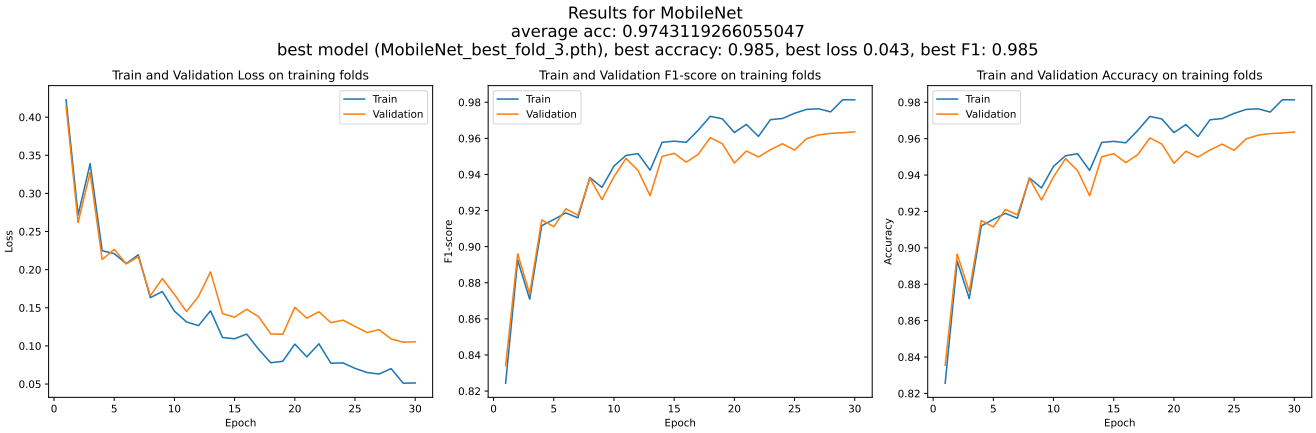

Figure 21: Average training and validation loss, F1-score and accuracy over all folds per epoch for the pre-trained MobileNetv2 3D model parameter settings 28

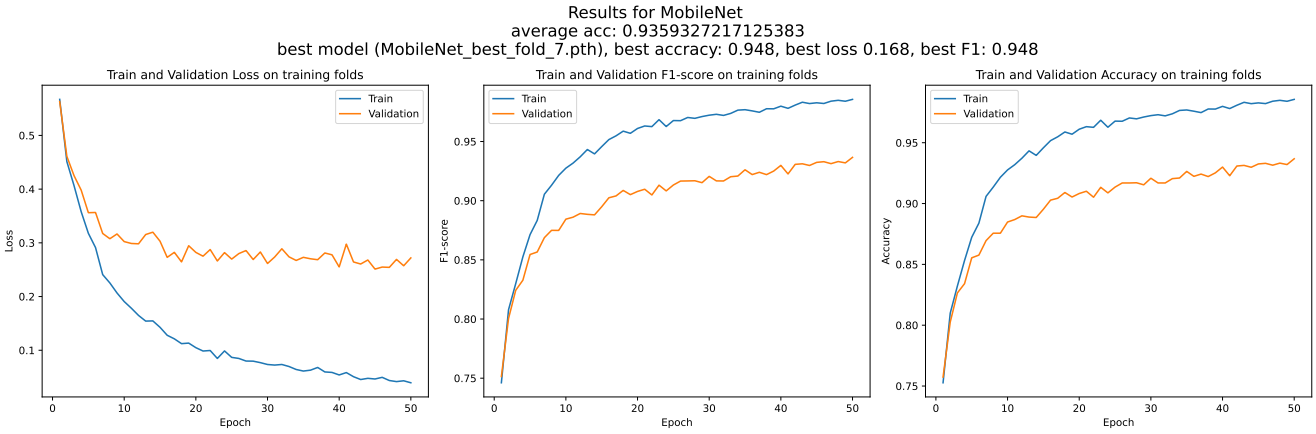

Figure 22: Performance graph of the MobileNetV2 model trained from scratch with 50 epochs on optimal parameters

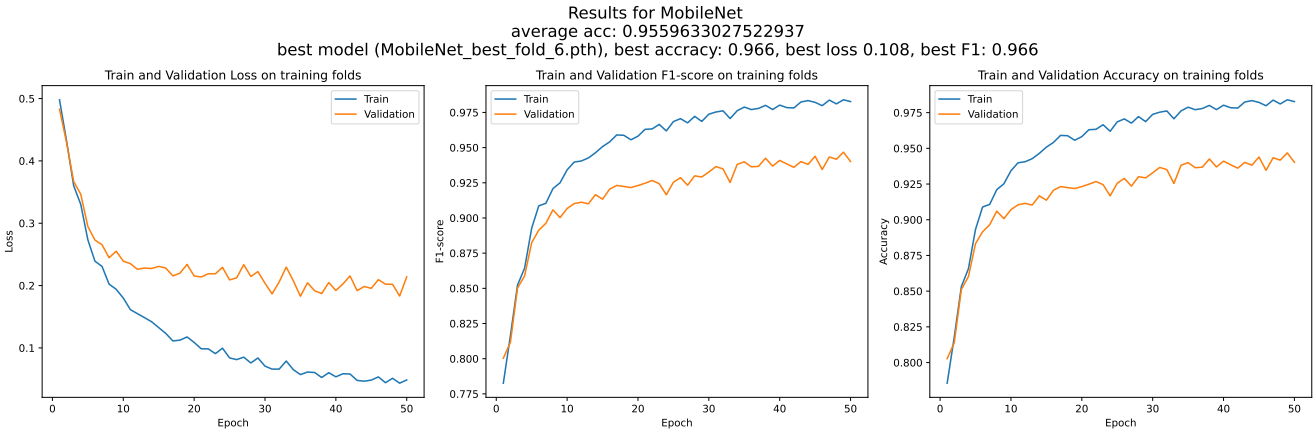

Figure 23: Performance graph of the pre-trained MobileNetV2 model trained from scratch with 50 epochs on optimal parameters