



Universiteit
Leiden
The Netherlands

Opleiding Informatica

Enhancing Pollen Classification Accuracy through the Use of Whole
Stack Images in 3D Convolutional Neural Networks

Shreya Sebastian

Supervisor:
Lu Cao

BACHELOR THESIS

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

23/08/2023

Abstract

Airborne pollen levels have escalated drastically due to the effects of climate change, hence the ability to forecast trends in the concentration of allergenic pollen has become increasingly crucial for proper management of allergic rhinitis and other respiratory conditions. There are two genera within the *Urticaceae* family called *Parietaria* and *Urtica*. Their pollen grains are structurally very similar but vary significantly in the level of allergy they induce, so being able to identify them is crucial but difficult to do manually. The aim of this paper is to investigate the potential improvement in the classification accuracy of pollen strains using raw stack images in a 3D convolutional neural network in place of 2D projections. Both the whole stacks as well as various ranges of central subsections of the stack were used for training as the peripheral frames of the stack are out of focus. Additionally, transfer learning was implemented using the pre-trained *r3d_18* model on the whole stack. The ResNet3D model trained on whole stack pollen images was the best-performing model among both the 3D CNN models evaluated in this paper, producing a classification accuracy of 95%.

Contents

1	Introduction	1
1.1	Literature Review	1
1.2	2D Image Projections	2
1.3	Raw Stack	3
1.3.1	Whole Stack	3
1.3.2	Central Subset of Stack	3
1.4	Transfer Learning	4
1.5	Overview	4
2	Convolutional Neural Networks	4
2.1	CNN Architecture	5
2.2	3D Convolutional Neural Networks	6
3	Methodology	6
3.1	Methodology Flowchart	6
3.2	Data Preprocessing	7
3.2.1	Padding	7
3.2.2	Data Augmentation	7
3.3	Method Overview	7
3.4	Software Libraries	8
3.4.1	Tensorflow/Keras	9
3.4.2	Pytorch	9
3.5	Project Models	9
3.5.1	Baseline ConvNet3D	9
3.5.2	ResNet3D	9
3.6	Training the Model	10
3.7	Testing the Model	10
4	Results	10
4.1	Baseline 3D CNN Model	10
4.2	ResNet3D Model	11
4.2.1	Augmentation Threshold	11
4.2.2	Learning Rate	12
4.2.3	Excluding Peripheral Frames	14
4.2.4	Batch Size	16
4.2.5	Pre-trained Model	17
5	Conclusion	19
6	Future Work	20
6.1	K-Fold Cross-Validation	20
6.2	Increasing the Dataset	20
	References	23

7	Appendix	24
7.1	A: File Structure	24
7.2	B: Standard Hyperparameters	24

1 Introduction

Pollen classification is an important task in several fields, such as botany, palynology and environmental sciences. Certain strains of pollen can result in allergic reactions and so it is important to be able to identify these specific strains by investigating differences in characteristics between pollen strains. Looking at a specific example, within the *Urticaceae* family are two genera, *Parietaria* and *Urtica* and these two pollen strains are morphologically quite similar yet induce allergic reactions to very different extents. The *Parietaria* strain can induce high levels of allergy in humans whereas *Urtica* is not nearly as harmful. Therefore, it is of interest to be able to forecast levels of each pollen strain in the air during a given season. This forecasting is especially relevant now as global warming and climate change have resulted in extended pollen seasons [Cip18]. To achieve this, it is essential to be able to differentiate between the pollen from these two genera in order to provide adequate safety regulations and guidelines for patients suffering from allergic rhinitis, commonly known as hay fever.

While it is common for pollen classification to be done simply under a microscope by a palynologist, manually distinguishing them is challenging when the pollen strains are so similar visually. The only species within the *Urticaceae* family that is distinguishable under a microscope is the *Urtica membranacea* species as it is notably smaller and has a higher number of pores than the other species. A solution to this is automatic classification with methods such as machine learning and deep learning-based techniques [Li23].

1.1 Literature Review

The paper “Analysis of automatic image classification methods for *Urticaceae* pollen classification” written by Chen Li et al., discusses the efficiency of both machine learning and deep learning-based methods for classifying *Parietaria* and *Urtica* pollen strains [Li23]. Deep learning-based methods were found to perform better, yielding an accuracy of 99.4% while the highest accuracy achieved by the machine learning methods was 94.5%. Therefore, the evaluation of the raw stack images will be done using deep learning techniques, specifically convolutional neural networks. The neural network architectures used were AlexNet, VGG16, VGG19, MobileNet V1, MobileNet V2 and ResNet50. The model ResNet50 was found to produce the best classification accuracy.

The dataset used for this paper consists of 6472 pollen stack images from which 2D projections were produced [Li23]. Whereas, the dataset for this paper is of a subset of that dataset, consisting of 600 3D pollen stack images. A pollen image stack consisted of 20 cross-sections along the Z-axis and the height x width dimensions varied for each individual pollen. The model will require that all of the input data have the same dimensions and so padding will be implemented in order to accomplish that.

There are several slices of the stack that are out-of-focus which could pose a problem for the classification model. To combat this, Z-stack projection methods were used to pre-process the images to create 3 2D projections of the whole stack. This was done to eliminate the potential problems the out-of-focus elements of the stack could cause during the learning of the deep learning models. While this does solve the problem of the blurry frames of the image stacks, there is information that is potentially lost.

The edges of the raw stack for this dataset are considerably noisy and this can interfere with the learning. Therefore, the data used as input for the neural networks in the paper are projections of the raw stack, specifically, three projections were produced and fed into the network (Standard Deviation Projection, Minimum Intensity Projection, Extend Focus Projection) [Li23].

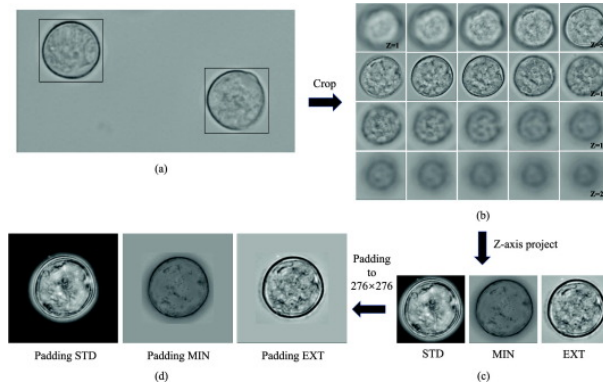


Figure 1: The workflow of pollen image projection acquisition from previous research. Source: [Li23].

Another paper, "Pollen Grains Recognition Based on Computer Vision Methods" by Amar Idrees Daood [Dao], introduced other methods for automatic pollen classification. Each of these methods catering to different types of image data: single-focus plane images from light microscopy (LM) and scanning electron microscopy (SEM) datasets. Another approach focused on multi-focal image sequences (Z-stack images), capturing a single pollen grain sample in various focal planes.

For single-focus images, a two-stage classifier was proposed. Texture features from the images were captured using the Leung-Malik filter bank, thereby forming an appearance model for pre-classification of the pollen into broad groups. The pollen images were segmented into multiple layers which enhanced this model, resulting in improved classification accuracy [Dao]. Additionally, in this study, convolutional and recurrent neural networks (CNNs and RNNs) were utilized to learn and classify pollen grains from stacks of multi-focal images. This method combines the capabilities of CNNs and RNNs to create a hybrid approach. The CNN learned optimal features while training on serialized multi-focal image datasets. Whereas, a RNN treated the multi-focal images as a sequential dataset. This approach in combination with transfer learning, achieved a remarkable 100% classification rate [Dao].

1.2 2D Image Projections

In terms of image processing, a projection is the transformation of a multi-dimensional image into a lower-dimensional image. For instance, converting a 3D pollen image stack to 2D image projections. Projections are used in order to extract specific information from the original image. Different types of projections highlight different aspects of the data.

The maximum intensity projection is common for 3D data. It projects maximum pixel intensity for each position, yielding a 2D image highlighting intense features. A common drawback as a result is low variation among the pixel values. The standard deviation projection computes the standard deviation of the pixel intensity across the stack. This reveals variability which is useful to assess regional pixel differences [Nog21]. For microscopy, extended focus projection merges images at various focal planes, creating a clear composite. This method is beneficial for thick specimens along the z-axis [FA08].

The 2D projections discard some of the depth information present in the raw stack, which can be integral for capturing the three-dimensional structural characteristics of pollen grains. Ignoring this information may limit the network’s ability to distinguish between complex and morphologically similar strains. These considerations raise the question of whether utilizing whole stack pollen images in a 3D convolutional neural network can outperform the use of projections of the stack in a 2D neural network for automatic pollen strain classification.

1.3 Raw Stack

In this paper, the input data will be the raw stack and so the convolutional neural network used will be a 3D neural network. 3D neural networks are able to capture features that a 2D neural network cannot but they are more computationally intensive in comparison. In this project, the Z-stack images will be evaluated without being pre-processed. The reasoning behind this is that raw stack images should theoretically have more information than projections.

1.3.1 Whole Stack

The whole stack is thought to possibly yield a higher classification accuracy due to its holistic nature. The blurry frames within the stack can pose a problem but keeping the stack as a whole has certain advantages. For example, using the full stack allows for spatial dependencies across the frames of the pollen images to be captured and this could potentially improve learning.

1.3.2 Central Subset of Stack

A solution for the noisy frames is to take just a subsection of the raw stack images. Specifically, taking middle subsections of varying sizes to reduce the number of frames that are out of focus can be done. This way, the frames that are used as input will be more in focus but there might still be important information in the edges of the stack which can be vital for aiding classification. Additionally, the positions at which the frames are out of focus can vary depending on the image in the dataset so choosing the range of the subsection to extract can be difficult.

The best-performing neural network was trained and tested on 3 ranges, namely the 5, 10 and 15 middle frames of the stack. While this method does indeed reduce the noise present in the data, it sacrifices the holistic nature of the whole stack and the advantages that come with it. Suppose the edges of the pollen grain are vital information for classification, then this method of data

preprocessing will not produce ideal results. The extracted subsection of the whole stack will still possess information about the spatial relationships between each of the frames present. However, the frames that are not present in the subsection are discarded entirely in contrast to using 2D projections where every frame is taken into account to a certain extent.

1.4 Transfer Learning

Transfer learning entails reusing a pre-trained model on a new dataset. This technique is especially useful when only a small amount of data is available for training. Essentially, the weights that a network has learned previously can be transferred to a new task. As a result of requiring less training data to achieve the same accuracy as a model that is not pre-trained, transfer learning can reduce training time drastically [Don19].

In this project, transfer learning was implemented in addition to using the same model without pre-trained weights.

1.5 Overview

Neither of the approaches discussed in the aforementioned papers employed a 3D convolutional neural network which is what will be done in this report. The second paper does however take the spatial relationships between each frame of the z-stack by treating them as sequential data and feeding it into a recurrent neural network. In this paper, raw pollen stack images are fed into 3D convolutional neural network models as opposed to 2D projections of the stacks. The hyperparameters for the models were fine-tuned based on the validation dataset accuracy. Both the whole stack as well as different subsections of the middle of the stack are used as input data for the models. The dataset was trained using both a custom baseline 3D convolutional neural network model as well as a ResNet3D model. Furthermore, the ResNet3D model was also trained using pre-trained weights.

2 Convolutional Neural Networks

Deep learning methods, namely neural networks are currently the best-performing solutions for classification tasks such as image and speech recognition and natural language processing [Nie15]. Convolutional neural networks (CNN) are a type of deep learning model designed specifically for classifying data, such as images and videos [CNN15].

These networks take inspiration from how the human visual system works and attempts to imitate the way the brain processes visual information [Kal21]. A CNN utilizes automatic feature extraction and hierarchical learning, and this makes it effective in computer vision tasks. The first layer of a neural network is the input layer and it is a representation of the data that the network will process. Specifically, in the context of images, the input layer consists of the image's pixel values. CNNs consist of several components that are unique to them, namely the convolutional layer, the

pooling layer and flattening [CNN15].

2.1 CNN Architecture

The convolutional layer is an integral part of CNNs because it performs convolution on the input data using filters called kernels. These filters slide over the input data, extracting features by computing element-wise products and summing them up. The outputs of this operation are feature maps, which focus on different elements of the input [Bro20b]. These layers are stacked sequentially to form a CNN architecture and layers that are close to the input layer are able to learn low-level features like lines in an input image. Whereas layers that are deeper in a network are able to learn high-order features like shapes or specific elements [Bro19].

The purpose of pooling layers is to reduce the spatial dimensions of the produced feature maps while still retaining essential information. The most common type of pooling used is max pooling and this is when the maximum value in the pooling window is selected and used as a representative value for a specific region. This helps reduce the computational complexity of the neural network [Bro19].

The fully connected layer, also known as a dense layer, is used to make predictions based on the features learned in previous layers. Each neuron in a fully connected layer is connected to every neuron in the previous layer. These layers make the final decision based on the features that are extracted by the previous layers [Bro20b]. Before feeding the output of previous layers into the fully connected layers, the feature maps are usually flattened into a 1D vector. This is done so that the input is compatible with the requirements of the fully connected layer [Tea18].

An activation function is applied to the output of a layer in a neural network. A common activation function used in CNNs is ReLU (Rectified Linear Unit) and this function's output is the input itself if it is positive and 0 otherwise. Softmax is another activation function and is used for multi-class classification problems. It returns the probability that a given input is in each class [?]. In this paper, the ReLU activation function is used within the hidden layers of the models and the Softmax activation function is used in the output layer to determine the class probabilities of each pollen image fed into the network.

The loss function is a function that returns a value that is representative of how well a model is able to predict classes for a given input dataset. The loss function does this by comparing both the target labels and the model's predicted labels and produces a measure of the deviation between these labels. A neural network model improves learning by adjusting its weights in order to minimize the loss value. The loss function used in this project is the categorical cross entropy loss function, which is suitable for multi-class classification problems as it measures the discrepancy between the predicted class probabilities and the target values [Yat22].

An optimizer is used to minimize the loss l during the training process by tuning the model parameters, namely the weights and biases of the model. The specific optimizer used in this project is the AdamW optimizer which is an improved version of the Adam optimizer (Adaptive Moment

Estimation). This optimizer is one of the most commonly used optimizers available and is a gradient descent optimization algorithm. Gradient descent algorithms work by iteratively adjusting the model's parameters so that the loss function approaches its local minimum [Mus22].

2.2 3D Convolutional Neural Networks

3D convolutional neural networks are best suited for working with 3D image data or videos. For video classification, each frame of the video over time is taken to produce a 3D stack of several image frames. 3D neural networks are able to capture the spatio-temporal dependencies that are needed for video classification. This pollen dataset consists of 3D image stacks for which a 3D convolutional neural network is required.

3D CNNs are suited for 3D data as a 3-dimensional kernel is used to perform convolutions of the image data. This kernel also moves in 3 directions due to the nature of the data and therefore produces a 3D output [Shi19].

3 Methodology

3.1 Methodology Flowchart

A flowchart describing the methodology of this project can be seen below:

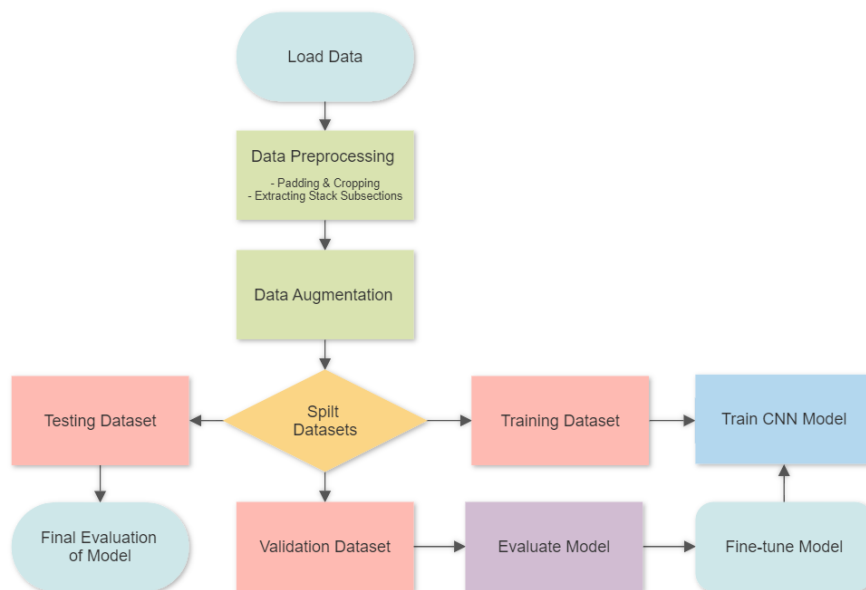


Figure 2: Flowchart of methodology describing data preprocessing, training and testing of the model.

3.2 Data Preprocessing

The data used in this project is a subset of the dataset used in the paper "Analysis of automatic image classification methods for Urticaceae pollen classification" written by Chen Li et al. The pollen image dataset was preprocessed in order to be compatible with the CNN models as well as aid learning.

3.2.1 Padding

The input for the classification models is required to be the same size but this was not the case for the dataset. This was solved by implementing zero padding on the pollen image dataset with the purpose of preserving information about the original sizes of the pollen grains, thereby retaining spatial information. Zero padding is simply padding the edges of the image array with zeros until a desired dimension is achieved [Dee18]. The alternative is resizing the images and this results in the loss of important information about size differences in the different pollen strains.

3.2.2 Data Augmentation

To compensate for the small size of the pollen dataset, augmentation was performed on the image data. A specified amount of the image dataset was selected for data augmentation. The transformations applied to the images included random vertical and horizontal flips as well as random rotation. This process artificially creates more data points which can aid the learning of the model. The amount of augmentation was also controlled through the use of a parameter because excessive amounts of augmented data can be counter-intuitive and can in fact interfere with learning.

3.3 Method Overview

Various Python libraries like numpy, pandas, torch, and others were imported for the purpose of handling data and for creating, training as well as evaluating the model. There are several hyperparameters and configuration settings for aspects of the code such as data preprocessing, data augmentation, training the model and so on.

The pollen image data paths were loaded and shuffled. The labels were extracted from each file path as the pollen species are organized into separate folders. Appendix A can be referred to in order to view the file structure of the project. The image data is padded, cropped and optionally a subsection of the frames can be extracted. The desired height and width after preprocessing are parameters that can be manipulated. For this project, the images are padded to 224 x 224 pixels since the majority of the image data dimensions fall below this threshold. The folder names are taken as the three pollen class labels : *P_judaica*, *P_of_ficinalis*, *U_dioica*, *U_urens* and *U_membranacea*. The class labels are used to map all the labels and convert them to integer labels.

The datasets that are created, contain the image, label and a boolean value representing whether the data point will be augmented. A custom dataset class was created to define how the data is loaded and augmented [Pyt23b]. The augmentation boolean value is read and if it is True the

transformations, horizontal flip and vertical flip are performed with a specified probability. A random rotation is also applied to the image with a specified maximum rotation angle. These transformed images are added to the training dataset as a means to artificially augment the input data. The image data is also normalized between pixel values of the range $[-1, 1]$ as this makes training the model more efficient.

A part of the data in the training dataset has the augmentation values set to True and the amount is based on a hyperparameter. In this manner, the amount of augmented data can be adjusted so that the optimum amount of augmentation can be established. The intent behind this is to achieve a balance to avoid overfitting or underfitting the training data. The augmentation threshold parameter is inversely proportional to the percentage increase of the dataset.

Data loaders are created using the resulting datasets from the custom dataset class. The purpose of these data loaders is to easily pass the dataset in mini-batches during training as well as shuffle the data if needed [Pyt23b]. For instance, it is desirable to shuffle the training dataset as that enhances learning. This is because the model does not learn the order of the data and so overfitting to the training data is avoided to an extent. Shuffling the training data results in the model being more generalized and robust [Pro23b].

The CNN model's performance for a given data loader can be evaluated. For each batch of data, the forward pass is executed through the model in order to get predictions in the form of logits. The loss is calculated using the categorical cross-entropy loss function on the targets and logits. The Pytorch function `nn.CrossEntropyLoss()` takes the logits as input instead of the predicted class labels [Pyt23a]. The F1 score and accuracy are then calculated as well and along with the loss, form the list of metrics that are used to evaluate each model in this paper.

A summary of the given model architecture, parameters and output shapes after each layer is produced during training. The model is first moved to the GPU so that training can be sped up considerably. The model is trained using the augmented training dataset and is then evaluated on both the training and validation sets. The chosen metrics (F1 score, accuracy and loss) are tracked for both the training and validation set at every epoch. The current best model is saved by means of tracking the best validation set accuracy thus far. Additionally, the results per epoch are saved so that graphs displaying the change in metric values over the epochs can be plotted.

The best-saved model is loaded from the saved checkpoint and is then evaluated on the test dataset. Again, the model is moved to the GPU and then set to evaluation mode. The test dataset metrics highlighting the model's performance are printed out.

3.4 Software Libraries

A Python framework for deep learning was chosen based on the requirements of the project such as the type of input data or the type of models that are needed. This framework was then used to create code to create, train and test 3D CNN models (either custom or imported) on the 3D pollen image dataset.

There are a range of open source software libraries however two of the most popular frameworks are Tensorflow and Pytorch. Both of these machine learning frameworks are powerful tools for the purpose of implementing and training deep learning models. However, they each possess their own strengths and weaknesses depending on the use case.

3.4.1 Tensorflow/Keras

Previous research investigating the classification of *Urtica* and *Parietaria* pollen strains using 2D projection made use of the Tensorflow Keras library [Li23]. This library is well suited to handle 2D data augmentation and has a range of 2D convolutional neural network models that can be imported with ease. However, there is no custom data augmentation function for 3-dimensional data. The library also does not possess any ready-made 3D CNN models.

3.4.2 Pytorch

The 3D convolutional neural network models discussed in this paper were made using the Pytorch library. This is due to the fact that Pytorch is better suited for 3D data augmentation and has 3D CNN models that can be imported. Data augmentation of the image dataset can be performed with relative ease using Datasets and Dataloaders imported from `torch.utils.data`.

3.5 Project Models

Two different models were trained and tested on the pollen image dataset. The first model is a baseline 3D CNN model made with the intent of serving as a simple reference for the ResNet3D model. The second model, ResNet3D, is an imported 3D CNN model from the Pytorch library. Both the pre-trained version as well as the non-pre-trained version of the model are trained on the pollen dataset.

3.5.1 Baseline ConvNet3D

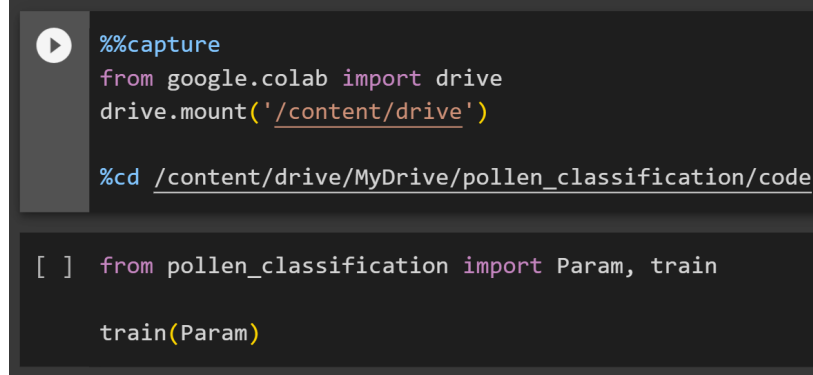
The Baseline ConvNet3D model is a simple 3D CNN model consisting of two convolutional layers, one max pooling layer, a dropout layer and finally two fully-connected layers. The ReLu activation function is applied within the hidden layers.

3.5.2 ResNet3D

The ResNet3D model used is the *r3d_18* model and is imported from the Pytorch library: `torchvision.models`. The model is an 18-layer 3-dimensional Resnet model and was used without pre-trained weights [Pyt19]. The best-performing model parameters were used again using the pre-trained version of the model. The ResNet50 model was used in previous research concerning *Urticaceae* pollen and was found to be the best-performing model [Li23]. For that reason, a 3D version of the ResNet model was implemented in this paper.

3.6 Training the Model

Due to the computational complexity of using a 3D convolutional network, the training time is extensive and therefore Google Colab was used due to its GPU capabilities. The GPU available in Google Colab is a Tesla T4 processor and was used to speed up training significantly.



```
%%capture
from google.colab import drive
drive.mount('/content/drive')

%cd /content/drive/MyDrive/pollen_classification/code

[ ] from pollen_classification import Param, train

train(Param)
```

Figure 3: Google Colab notebook containing pollen classification trainer code.

The *pollen_classification* folder contains the code folder and is imported from Google Drive. In the next code block, the `train()` function is called using the specified parameters in the `Param` class.

3.7 Testing the Model

The accuracy of the saved model can then be tested using the `test()` function. As this is a much smaller dataset and several epochs are not required, Google Colab's GPU is not necessarily needed.

4 Results

The results for both 3D CNN models: ConvNet3D and ResNet3D are discussed in the sections below.

4.1 Baseline 3D CNN Model

The first model that was trained and tested was the baseline 3D convolutional neural network: ConvNet3D. This was done using the standard parameters and the whole stack, therefore an image stack containing 20 frames. The standard hyperparameters within the `Param` class can be referred to in Appendix B.

After plotting each of the test metrics over the epochs, the large difference between the training and validation sets can be seen. The model continues to learn on the training set while the learning stabilizes just after the 5th epoch for the validation set.

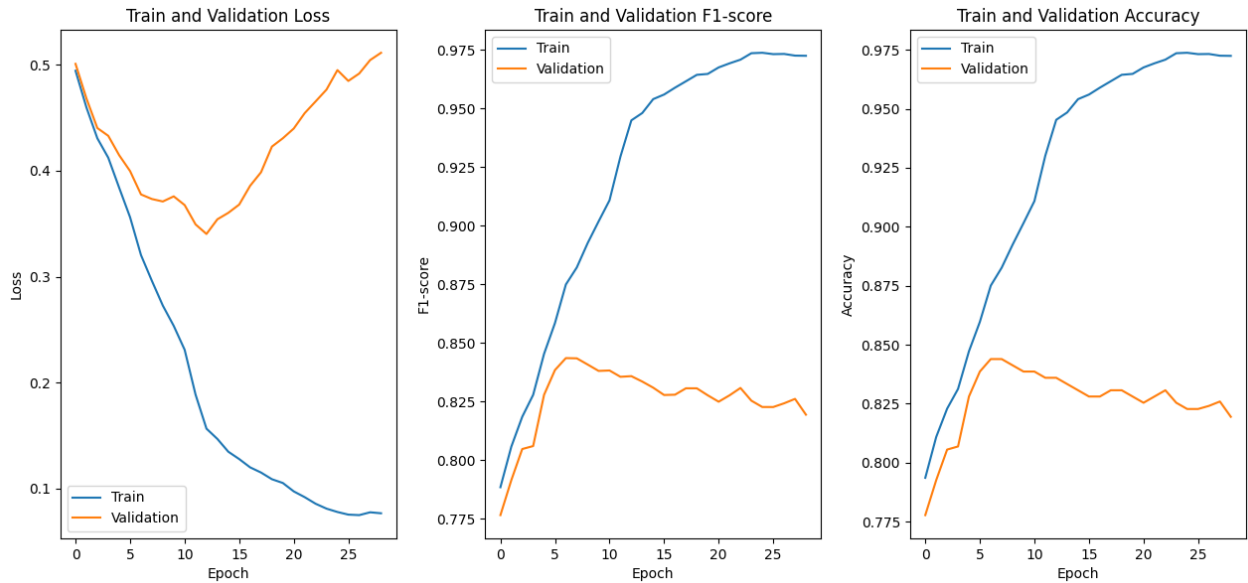


Figure 4: ConvNet3D model test metrics plotted, tested on whole stack

Furthermore, the test accuracy achieved by the baseline model can certainly be improved. The test metrics can be seen in the table below.

ConvNet3D	loss	f1 score	accuracy
20 frames	0.473	0.807	0.817

Table 1: ConvNet3D model test metrics, tested on whole stack

4.2 ResNet3D Model

The rest of the experiments were conducted using the 3D ResNet model *r3d_18*. The hyperparameters were fine-tuned based on the validation dataset accuracy. Namely, the augmentation threshold, learning rate and batch size were adjusted to find the optimal parameters for the model.

4.2.1 Augmentation Threshold

One of the first parameters that was tuned was the augmentation threshold parameter, representing the amount of augmented data that would be added to enhance the training dataset. The parameter value approximately equals the percentage of the training data set that is not taken for augmentation. Below is the table of test accuracy produced from a ResNet3D model trained on a whole stack for a short 3 epochs.

aug_threshold	test accuracy
0	0.70
0.25	0.71
0.5	0.79
0.75	0.70

Table 2: ResNet3D test accuracy after 3 epochs, tested on varying *aug_threshold* values

4.2.2 Learning Rate

Three learning rate values were tested using the whole stack image data. The learning curve for each of the test metrics can be seen in the graphs below.



Figure 5: ResNet3D model test metrics for learning rate:1e-4 plotted, tested on whole stack

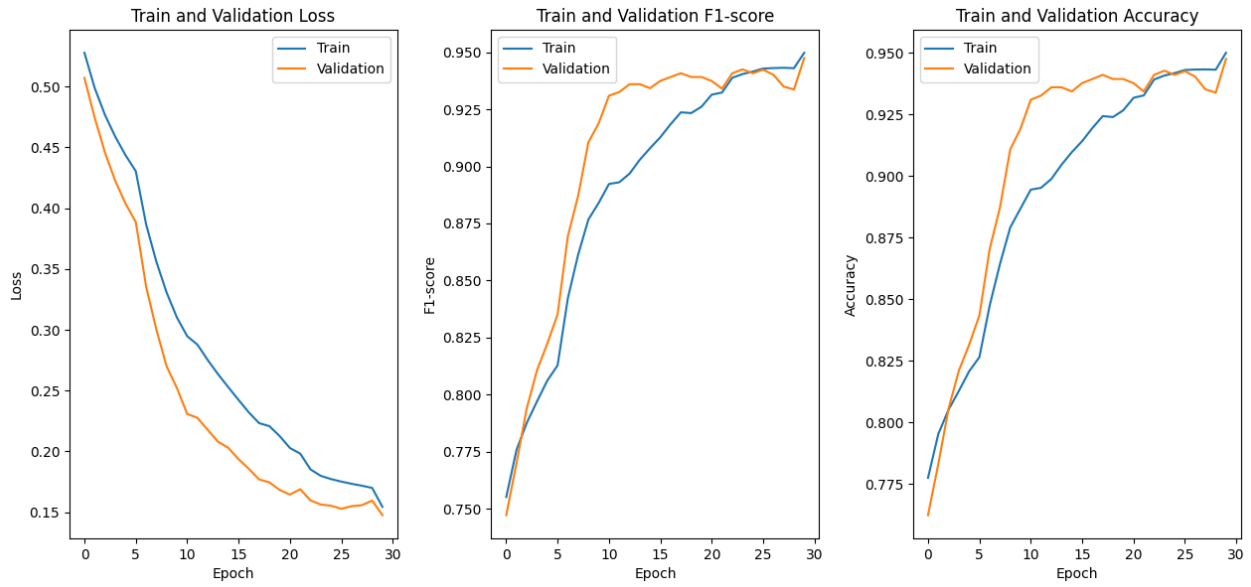


Figure 6: ResNet3D model test metrics for learning rate:1e-5 plotted, tested on whole stack

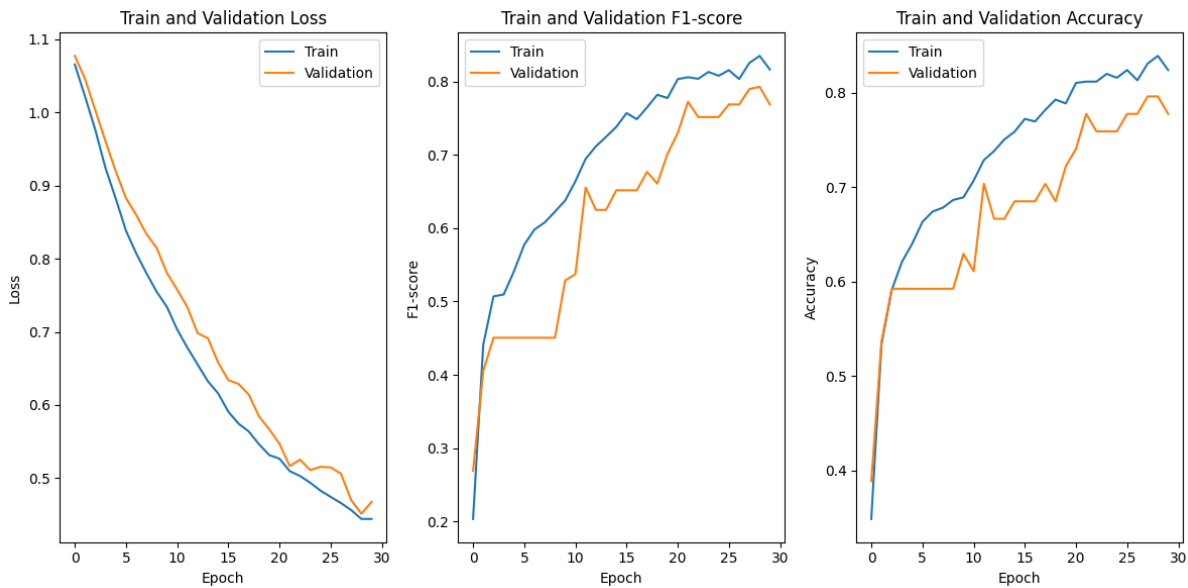


Figure 7: ResNet3D model test metrics for learning rate:1e-6 plotted, tested on whole stack

The learning curves produced by the ResNet3D model with learning rates $1e-4$ and $1e-5$ performed marginally better than using a learning rate of $1e-6$.

ResNet3D	loss	f1 score	accuracy
lr: 1e-4	0.211	0.949	0.950
lr: 1e-5	0.363	0.859	0.867
lr: 1e-6	0.444	0.788	0.800

Table 3: ResNet3D test metrics for various learning rates, tested on whole stack

The best performing learning rate was 1e-4 as it produced the highest classification accuracy with the standard parameter settings 3.

4.2.3 Excluding Peripheral Frames

Experiments were conducted using image stack data where the outer frames of the stack were discarded due to being out of focus. Various ranges of the middle frames were extracted and fed into the ResNet3D model. Specifically, the middle 5, 10 and 15 frames were selected and tested alongside the whole stack. The standard hyperparameters are used apart from n_frames .

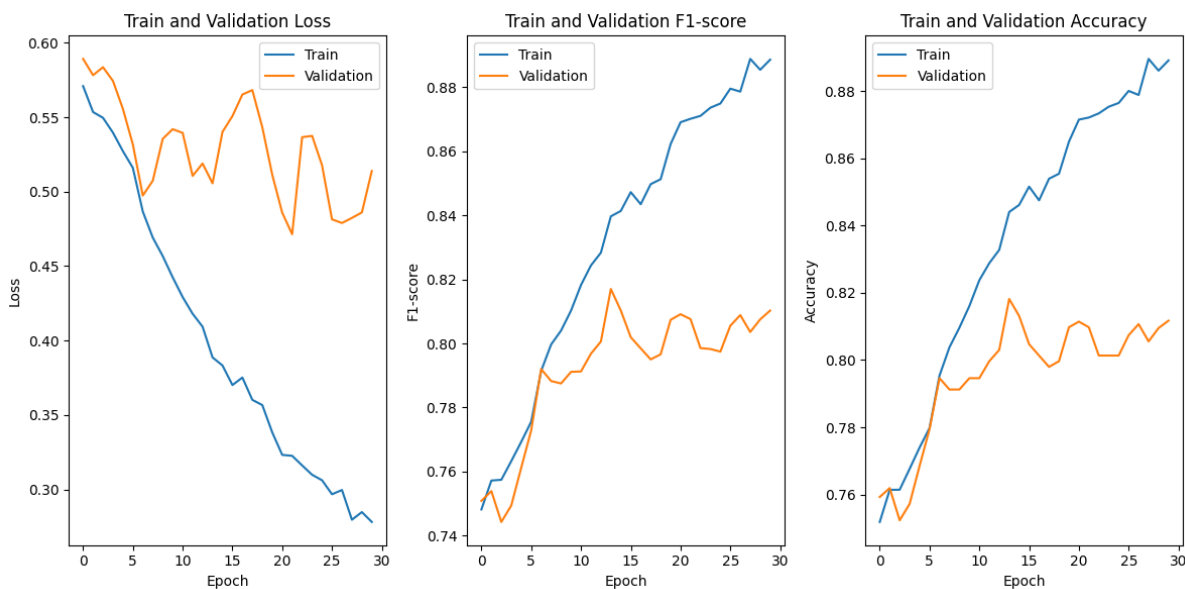


Figure 8: ResNet3D model test metrics plotted, tested on middle 5 frames

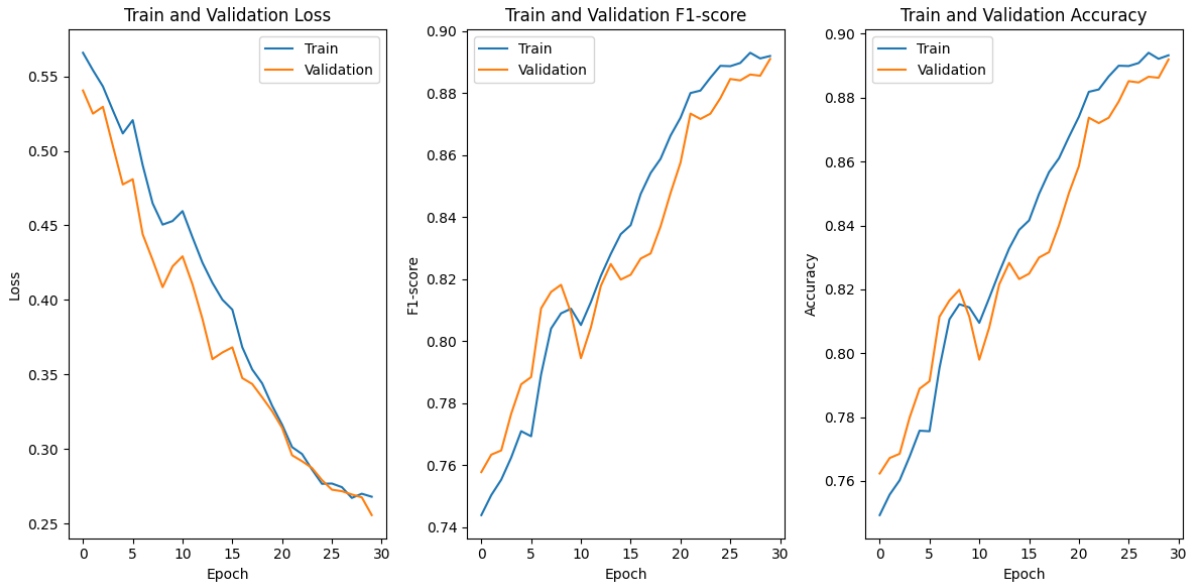


Figure 9: ResNet3D model test metrics plotted, tested on middle 10 frames



Figure 10: ResNet3D model test metrics plotted, tested on middle 15 frames

The results from using the middle frames show that the learning is substantially more unstable in comparison to the whole stack.

ResNet3D	loss	f1 score	accuracy
5 frames	0.318	0.883	0.883
10 frames	0.403	0.882	0.883
15 frames	0.369	0.879	0.883
20 frames	0.211	0.948	0.950

Table 4: ResNet3D model test metrics, tested on various ranges of middle frames

Based on the test dataset accuracies, the best performing ResNet3D model was the model that was trained on the whole stack.

4.2.4 Batch Size

The train batch size was increased from 16 to 32 to investigate whether a larger batch size improves learning.

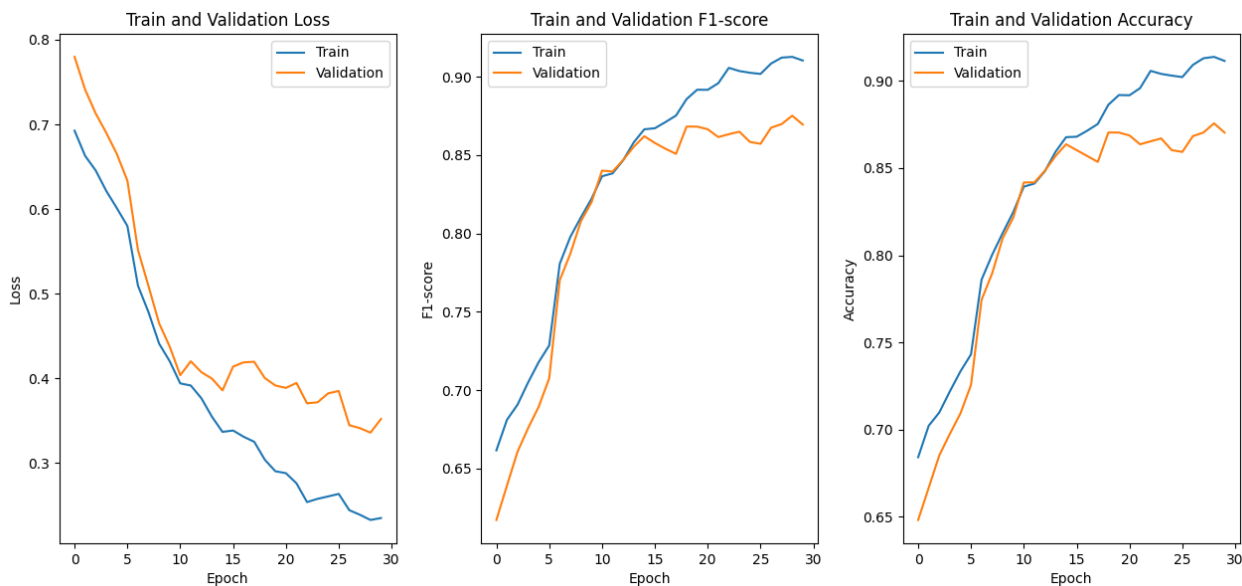


Figure 11: ResNet3D model test metrics for batch size: 32 plotted, tested on middle 15 frames

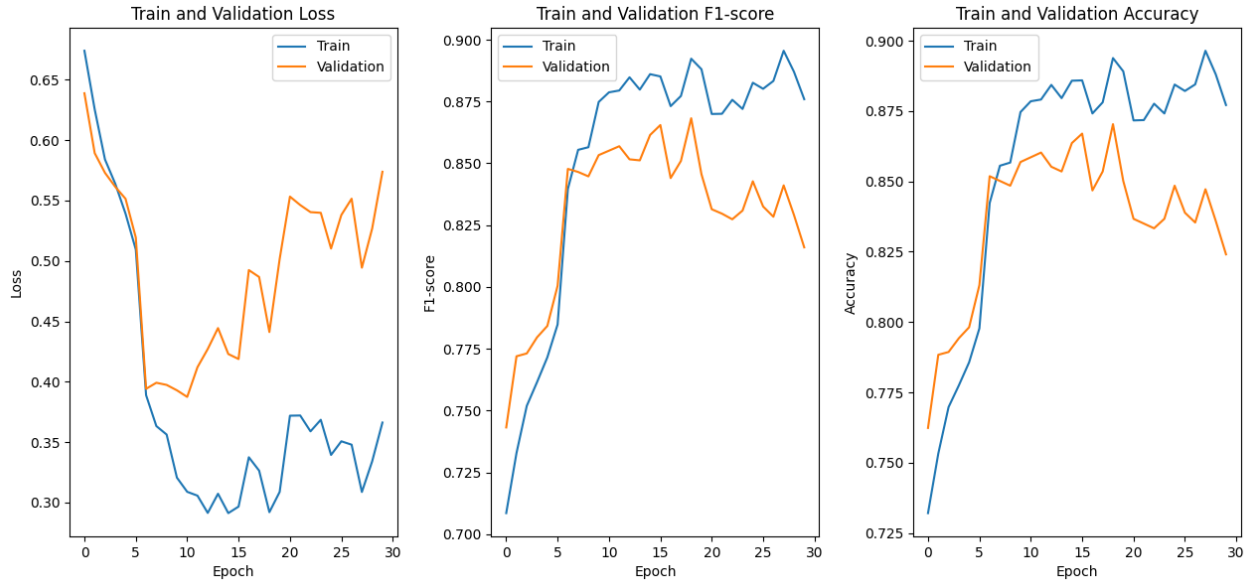


Figure 12: ResNet3D model test metrics for batch size: 32 plotted, tested on whole stack

The classification accuracy did not improve for either the 15 frames subsection or the whole stack (20 frames). The accuracy value for a batch size of 16 trained and tested with the whole stack was 95%.

ResNet3D	loss	f1 score	accuracy
15 frames	0.513	0.823	0.833
20 frames	0.237	0.918	0.917

Table 5: ResNet3D test metrics for train batch size: 32, tested on 15 and 20 middle frames

4.2.5 Pre-trained Model

After fine-tuning the hyperparameters and determining which range of image cross-sections is optimal for learning, the best model was trained once more using transfer learning. The pre-trained version of the $r3d_18$ was utilized.

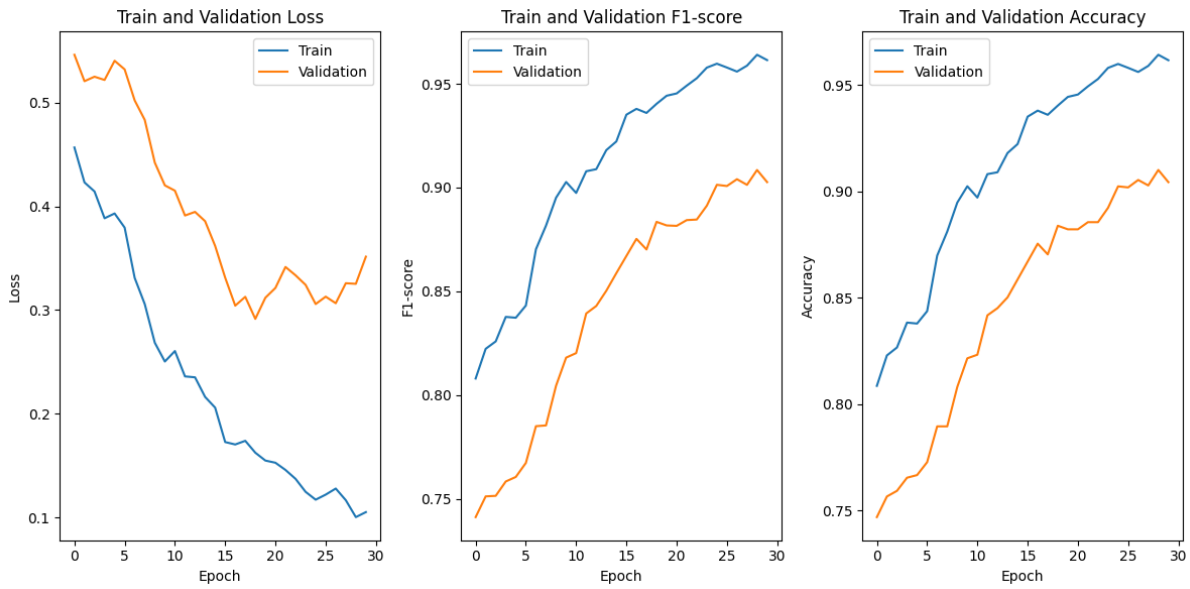


Figure 13: pre-trained ResNet3D model test metrics plotted, tested on whole stack

The pre-trained ResNet3D model showed an upward trend even up to 30 epochs. Due to the fact that the learning curve did not stagnate, another version of the same model was trained using 50 epochs.

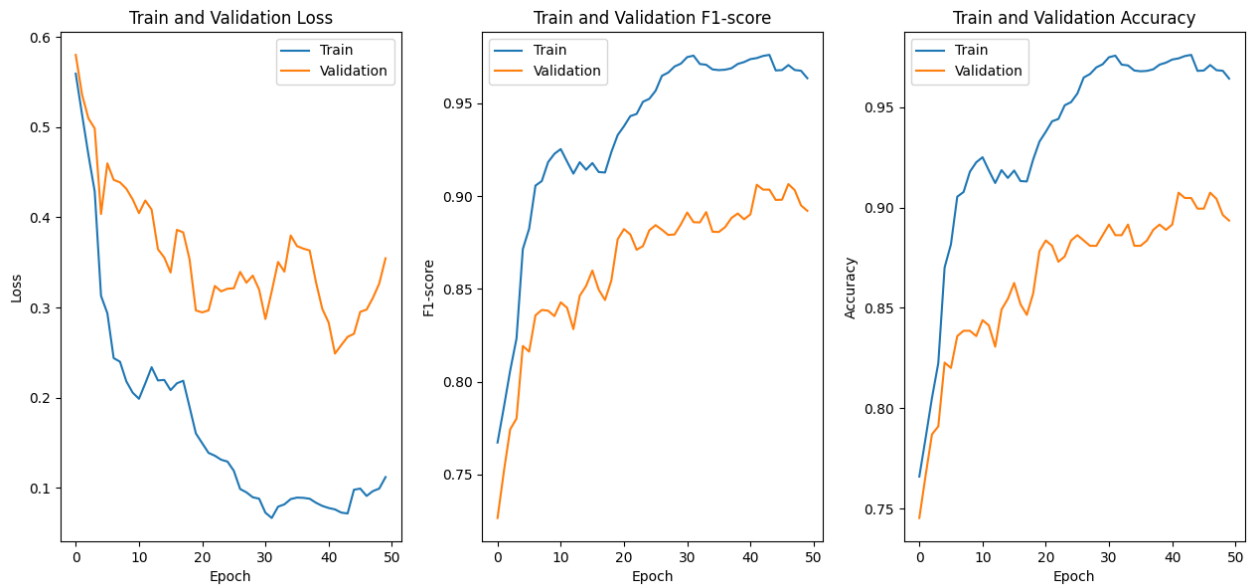


Figure 14: pre-trained ResNet3D model test metrics for 50 epochs plotted, tested on whole stack

The accuracy of the pre-trained model continued to improve after 30 epochs as expected. However, despite training for 50 epochs the validation learning curve did not show signs of stagnation.

ResNet3D	loss	f1 score	accuracy
30 epochs	0.456	0.865	0.867
50 epochs	0.311	0.878	0.883

Table 6: ResNet3D model test metrics, tested on various ranges of middle frames

Unexpectedly, the classification accuracies for the test dataset from the pre-trained ResNet3D models are marginally lower than their non pre-trained counterpart.

5 Conclusion

The blurry frames within the stack were expected to serve as a bottleneck for this project however it affected learning less than expected. Training the model on subsections of the stack resulted in the learning curve being rather unstable and the test accuracy for each of these subsections decreased as a result of that. Each of these subsections were trained with the standard hyperparameters alongside the whole stack (20 frames). The whole stack being fed into the ResNet3D model resulted in the best metrics values as well as showed the most stable learning curve. This model produced a classification accuracy that was marginally better than all of the models trained on the subsections. This indicates that the holistic nature of the whole stack is essential for proper learning. If the blurry frames affect learning, then it is not to the extent that excluding them and using a subset would improve learning.

The models trained on higher batch sizes were not more accurate despite having more data for training. This is because the model is overfitting on the training data which can be seen in figures 11 and 12. Decreasing the batch size can improve the accuracy of the model since it becomes more robust and can generalize better to unseen data [KM20].

Some graphs have validation accuracies that are higher than the training accuracies which can be explained by the use of data augmentation. Applying data augmentation during training can introduce some level of noise to the training data, potentially making it more challenging for the model to accurately predict the classes for the augmented training data. On the other hand, the validation set does not contain any augmented data and this makes it easier for the model to predict the classes for these images [Pro23a].

A possible reason for why the classification accuracies stabilized at the values that they did could be that the dataset was not large enough. Another reason could be that the hyperparameters were not optimal. The complexity of a 3-dimensional CNN model results in the fine-tuning of the hyperparameters being more challenging. 3D convolution operations entail far more parameters than 2D convolutions [Yan19]. Additionally, 3D CNNs require more training data than their 2D counterparts to achieve the same level of accuracy. Deeper networks do not have the ability to generalise as well

to new data since this added complexity results in the model being prone to overfitting on the training data, making it vital that the dataset is large and diverse enough to mitigate this risk [Sin20].

Utilizing transfer learning did not improve the accuracy as one would expect, instead the accuracies are distinctly lower. This can be seen in figures 13 and 14. A possible explanation for this could have been that the pre-trained weights of this model did not transfer well to the pollen image data however the training accuracy is higher in the pre-trained models than any other model in this report. Therefore it can be concluded that the model is overfitting on the training dataset. This can happen when the data available is not sufficient, causing the model to memorize rather than learn. In this manner, the model cannot generalize as well to new data and as a result the classification accuracy is lower [VC21].

The highest accuracy achieved in this paper was produced by the non pre-trained ResNet3D model trained on the whole stack image dataset. This model resulted in an accuracy of 95% which is not higher than the accuracies produced in previous research conducted on 2D neural networks. There may be potential for the model to achieve a higher accuracy with more fine-tuning of the hyperparameters or training it on a larger dataset.

6 Future Work

This research can be extended in several ways in order to achieve a better-performing 3D CNN model. A more complex custom 3D CNN model or additional Pytorch 3D CNN models could be used for training. The main obstacle in this paper was likely the lack of pollen image data, so the most effective strategies to combat that specific problem are mentioned below.

6.1 K-Fold Cross-Validation

The size of the test dataset in this paper is 10% of the total pollen image dataset (60 whole stack images). By implementing K-fold cross-validation, the model can be properly evaluated on multiple validation sets, thereby providing a more accurate estimate of its performance on unseen data. In this paper, the test and validations may not sufficiently large enough to provide a proper estimate so this technique would benefit the evaluation of the models greatly. K-fold cross-validation is a technique commonly used to more efficiently assess a model's performance and its generalization ability. [Bro20a].

6.2 Increasing the Dataset

The simplest solution to address the challenge of limited pollen image data is to expand the dataset. A larger subset of data could be taken from the Z-stack pollen image dataset present in the paper by Chen Li et al. [Li23]. The diversity of the dataset could also be intentionally increased by collecting and adding more pollen images to the training dataset, which can significantly enhance the model's ability to accurately classify pollen strains from the Uriticaceae family. By including more pollen

images, the model can learn to generalize better to different input data. It is difficult to estimate exactly how large the data set should be as this depends on aspects such as the model's complexity. A drawback to this method is that the training time for a 3D CNN model will be extended.

References

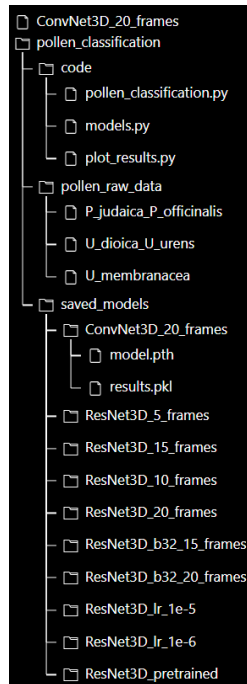
- [Bro19] Jason. Brownlee. A gentle introduction to pooling layers for convolutional neural networks, 2019.
- [Bro20a] Jason Brownlee. A gentle introduction to k-fold cross-validation, 2020.
- [Bro20b] Jason. Brownlee. How do convolutional layers work in deep learning neural networks?, 2020.
- [Cip18] et al. Ciprandi, Giorgio. Parietaria allergy: An intriguing challenge for the allergist. 54:106, 2018.
- [CNN15] What are convolutional neural networks?, 2015.
- [Dao] Amar Idrees Daood. Pollen grains recognition based on computer vision methods.
- [Dee18] Deeplizard. Zero padding in convolutional neural networks explained, 2018.
- [Don19] Niklas. Donges. What is transfer learning? exploring the popular deep learning approach, 2019.
- [FA08] M. Unser F. Aguet, D. Van De Ville. Model-based 2.5-d deconvolution for extended depth of field in brightfield microscopy. *Scientific Programming*, page 1144–1153, 2008.
- [Kal21] Gopal Kalpande. Biological inspiration of convolutional neural network (cnn), 2021.
- [KM20] Ibrahem Kandel and Castelli Mauro. The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. 6:312–15, 2020.
- [Li23] et al Li, Chen. Analysis of automatic image classification methods for urticaceae pollen classification. *Neurocomputing*, 522:181–93, 2023.
- [Mus22] Mustafa. Optimizers in deep learning - mlearning.ai - medium, 2022.
- [Nie15] Michael A. Nielsen. Neural networks and deep learning, 2015.
- [Nog21] Damian Dalle Nogare. An introduction to biological image processing in imagej, part 3: Stacks and stack projections, 2021.
- [Pro23a] Lazy. Programmer. What is higher validation accuracy than training accuracy in tensorflow and keras, 2023.
- [Pro23b] Lazy. Programmer. Why shuffle data when doing stochastic gradient descent (sgd) and mini-batch gradient descent?, 2023.
- [Pyt19] Pytorch. Vision/torchvision/models/video/resnet.py at main · pytorch/vision, 2019.
- [Pyt23a] Pytorch. Crossentropyloss — pytorch 2.0 documentation, 2023.
- [Pyt23b] Pytorch. Datasets and dataloaders — pytorch tutorials 2.0.1+cu117 documentation, 2023.

- [Shi19] Shivambu. 3d convolutions: Understanding + usecase, 2019.
- [Sin20] et al. Singh, Satya P. 3d deep learning on medical images: A review. *Sensors*, 20, 2020.
- [Tea18] SuperDataScience Team. Convolutional neural networks (cnn): Step 3 - flattening, 2018.
- [VC21] Chris Von Csefalvay. Transfer learning: The dos and don'ts - starschema blog - medium, 2021.
- [Yan19] et al. Yang, Hao. Asymmetric 3d convolutional neural networks for action recognition. *Pattern Recognition*, 85:1–12, 2019.
- [Yat22] Vishal. Yathish. Loss functions and their use in neural networks - towards data science, 2022.

7 Appendix

7.1 A: File Structure

Below is the file structure of the project, each saved model has a model.pth file and a results.pkl file as seen for the folder *ConvNet3D_20_frames*.



7.2 B: Standard Hyperparameters

```
# Preprocessing parameters
height = 224
width = 224

# Augmentation parameters
aug_threshold = 0.5
max_rotation_angle = 100

# Train parameters
train_batch_size = 16
valid_test_batch_size = 16
learning_rate = 0.0001
num_epochs = 30

# Model parameters
in_channels = 20
```