

# Report 4

## Plant Disease Classification

Authors:

1. Krzysztof Nazar, 184698
2. Hubert Nacmer, 184546
3. Łukasz Kawa, 184948

GitHub: [link](#)

### 1. Methods and metrics used for testing

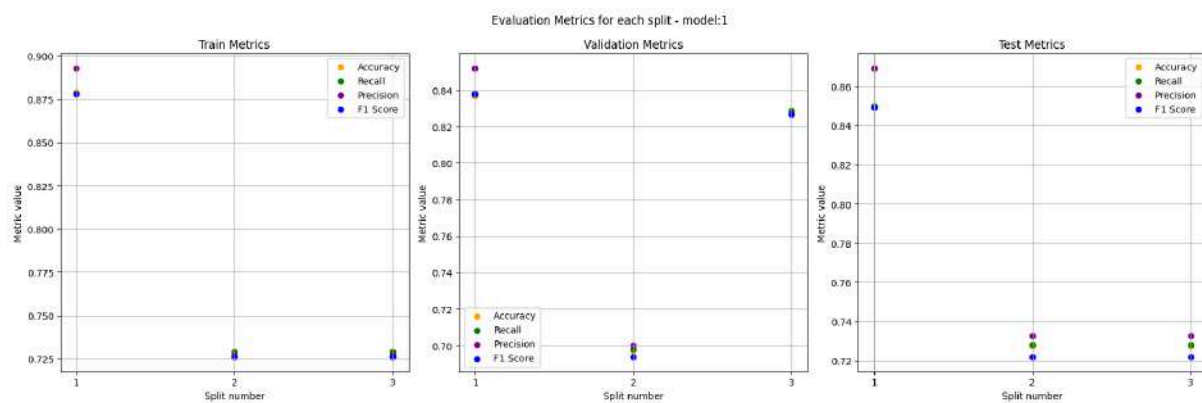
In order to evaluate the performance of the three trained models, we are going to use:

- Accuracy: helps to assess the general correctness of the classification results - it represents the ratio of correctly classified instances to the total number of instances.
- Precision: helps to measure the model's ability to minimise false positive predictions, ensuring that identified categories are indeed relevant - it represents the proportion of true positive predictions among all positive predictions made by the model.
- Recall: helps evaluate how effectively the model captures all instances of the target categories - it represents the proportion of true positive predictions among all actual positive instances in the dataset.
- F-score: helps to comprehensively evaluate the model's performance, considering both precision and recall simultaneously - it is the harmonic mean of precision and recall.
- Confusion Matrices - they offer insights into the types and frequencies of classification errors, which helps to understand the model's strengths and weaknesses across different classes. They are intuitive and relatively easy to read, understand, and gather meaningful conclusions.

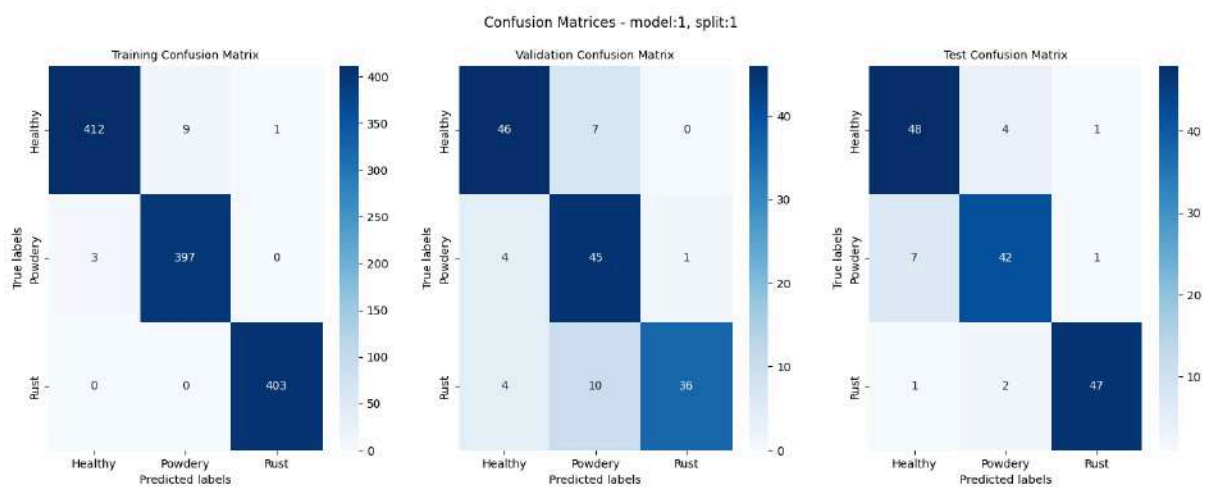
## 2. Testing the models

### 2.1. Model 1

#### 2.1.1. Evaluation Metrics

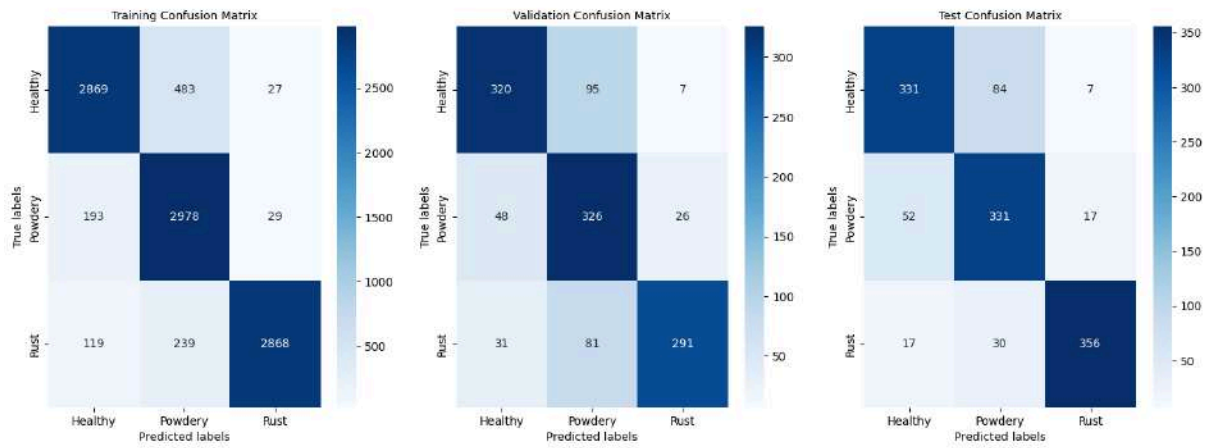


#### 2.1.2. Confusion Matrices

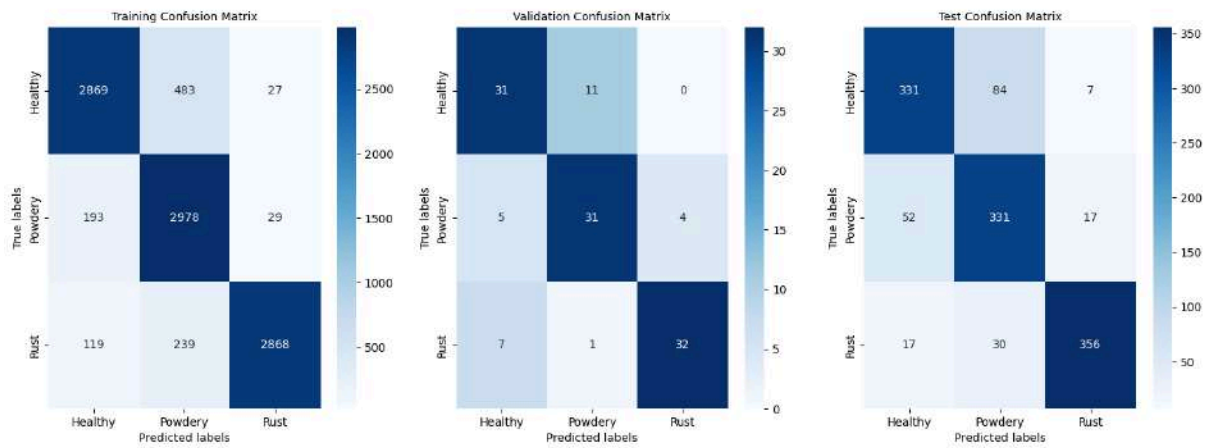


## SYSTEMS WITH MACHINE LEARNING - Task 4

Confusion Matrices - model:1, split:2

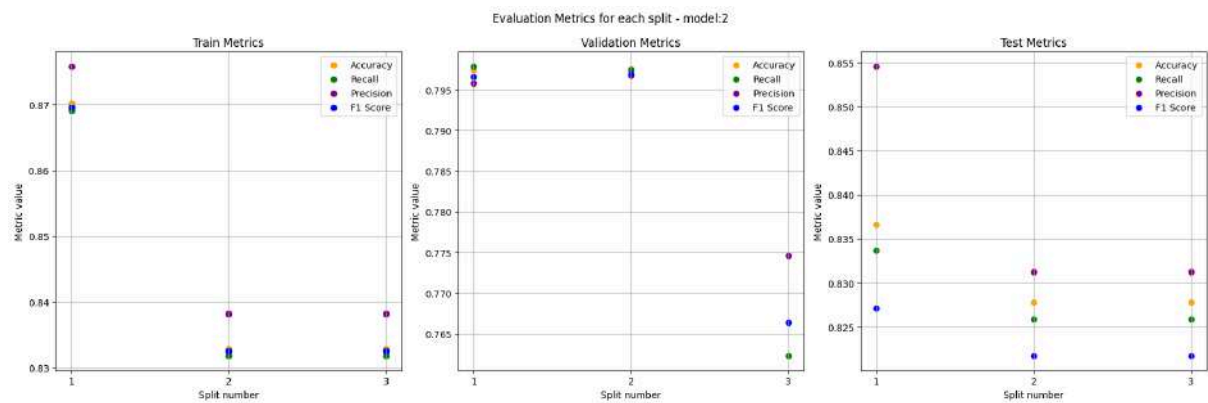


Confusion Matrices - model:1, split:3

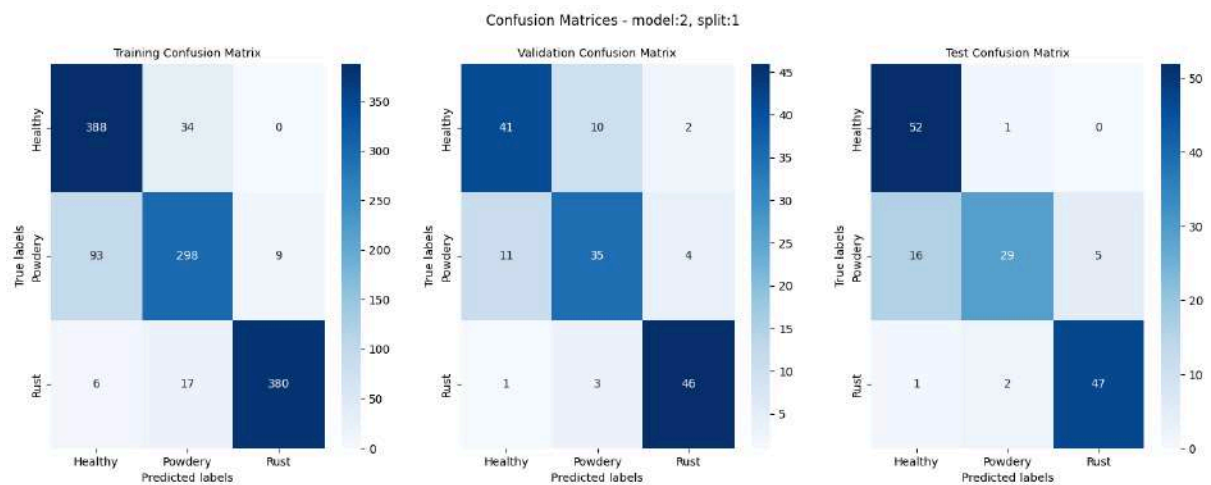


## 2.2. Model 2

### 2.2.1. Evaluation Metrics

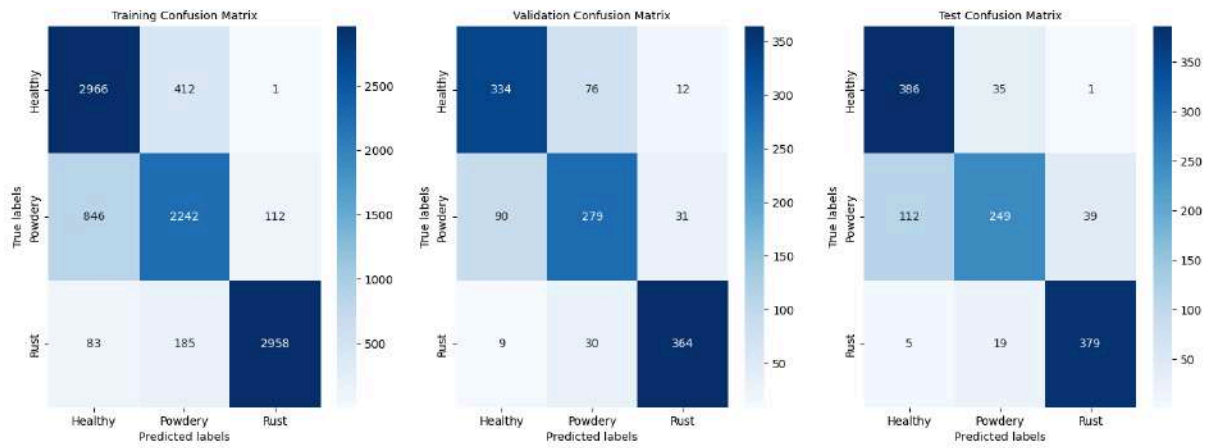


### 2.2.2. Confusion Matrices

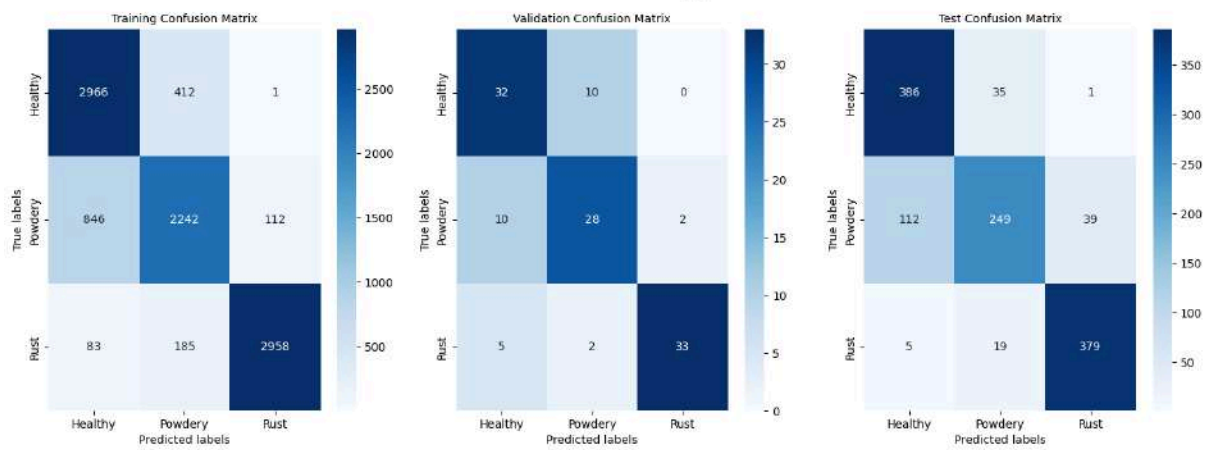


## SYSTEMS WITH MACHINE LEARNING - Task 4

Confusion Matrices - model:2, split:2

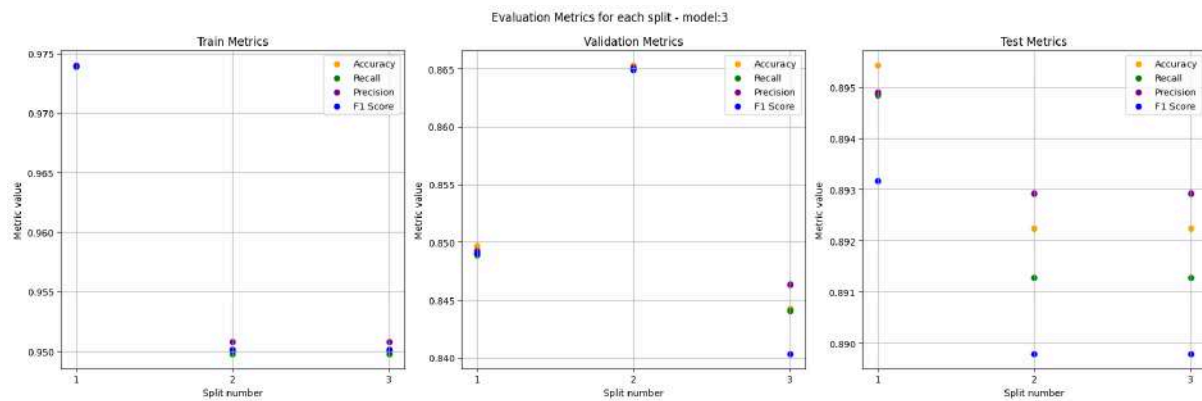


Confusion Matrices - model:2, split:3

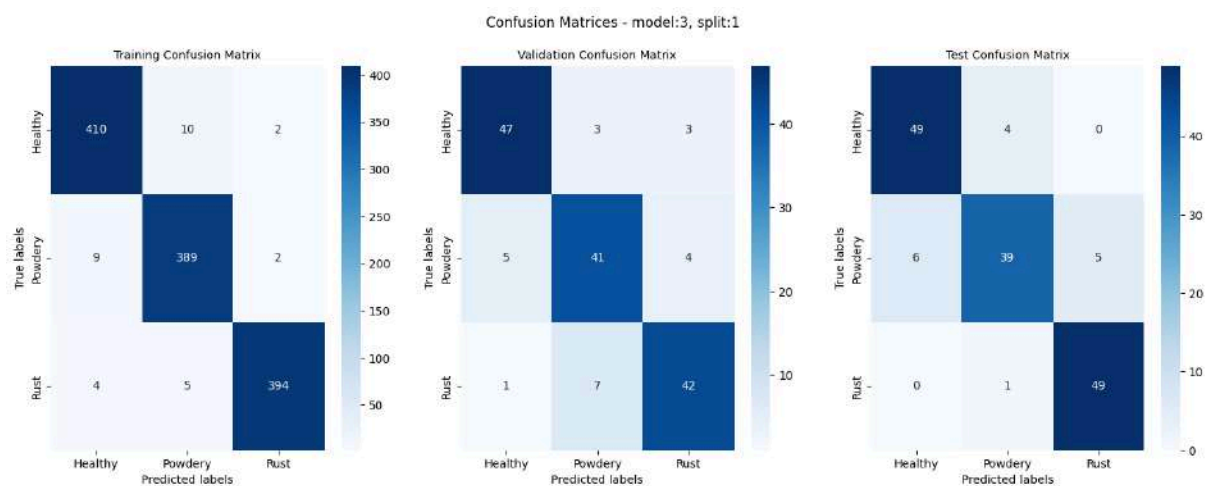


## 2.3. Model 3

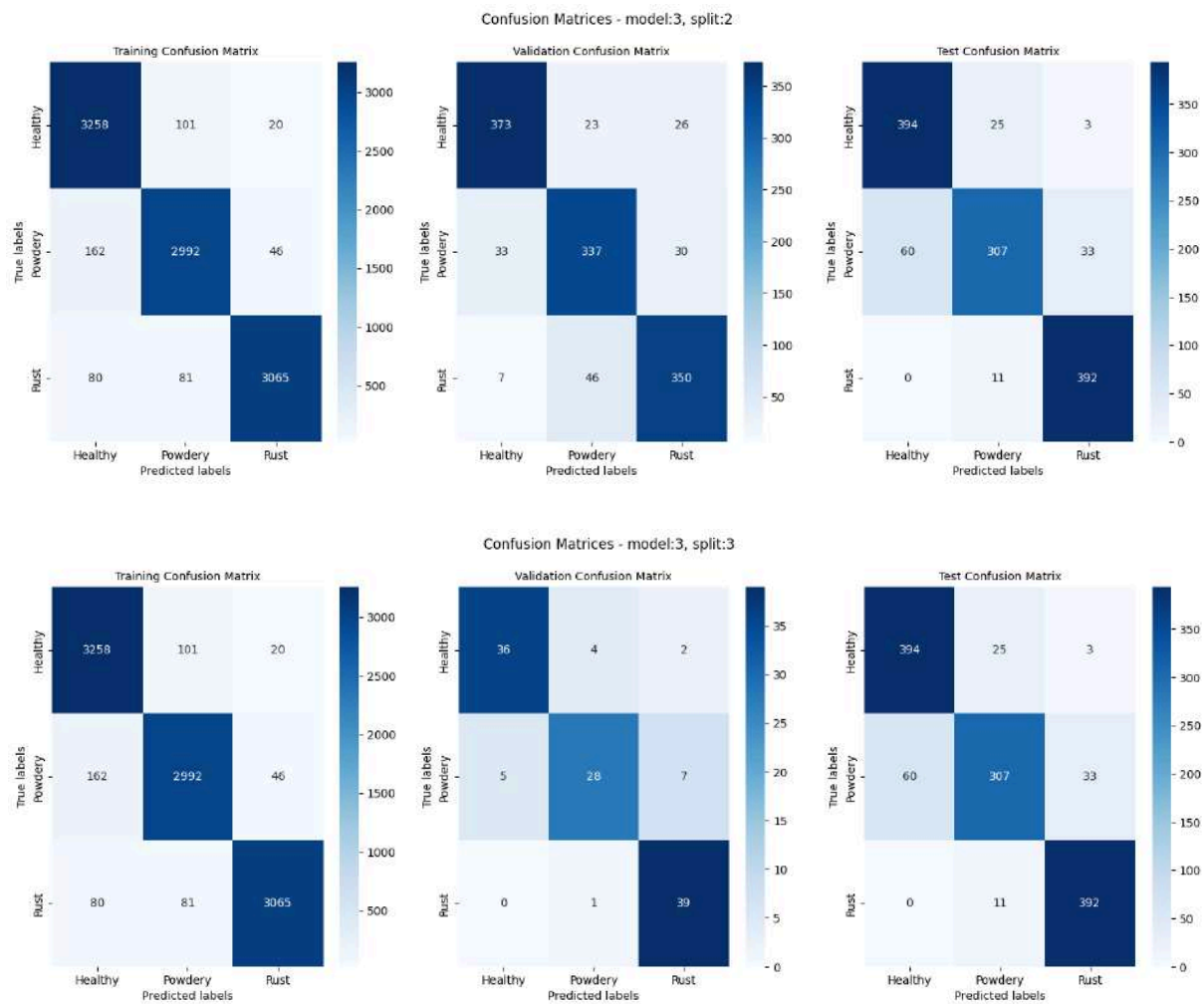
### 2.3.1. Evaluation Metrics



### 2.3.2. Confusion Matrices

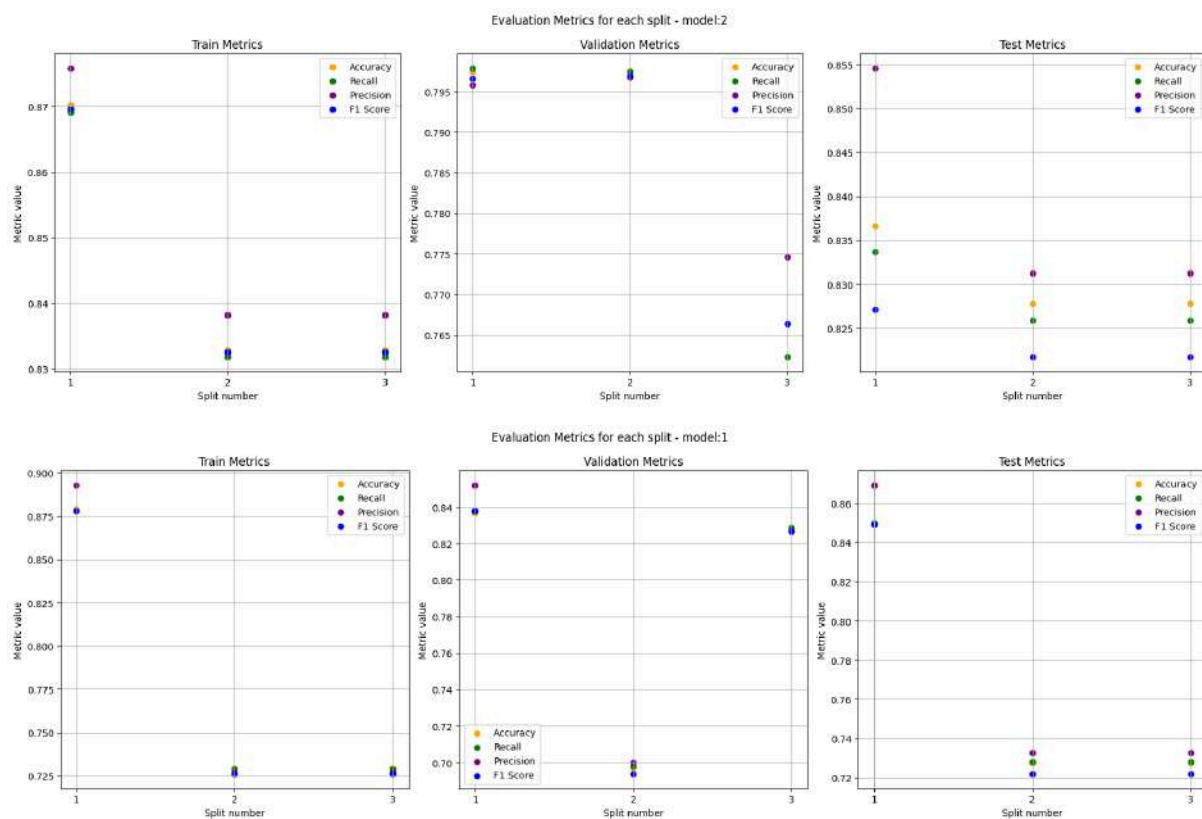


## SYSTEMS WITH MACHINE LEARNING - Task 4



### 2.4. Observations

The metrics reach their highest values in the third model. This could be due to having copies of some training samples in the validation samples (validation set use part of training set, keeping training and test the same), leading to the model being evaluated on the same images it was trained on. The results in the second model are less varied than in the first model (the standard deviation of values is smaller in the second model). This is likely because there is a larger amount of data used to train the second model - if we take the average metrics values, we can see that the second model is more accurate.



## 3. Model Improvement

For improving we chose the model 2 trained on the split 2, because the training data seemed to have had the best distribution between training, validation and testing subsets.

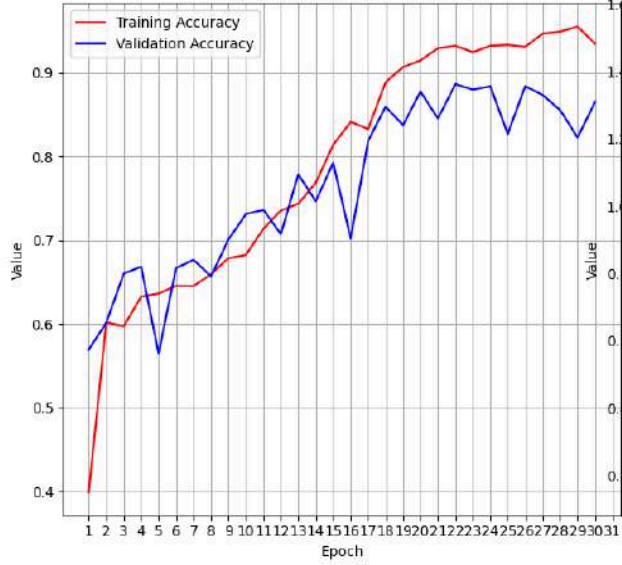
### 3.1. Increasing Epochs

First improvement of the model was to increase the number of epochs of training, from 15 to 30.

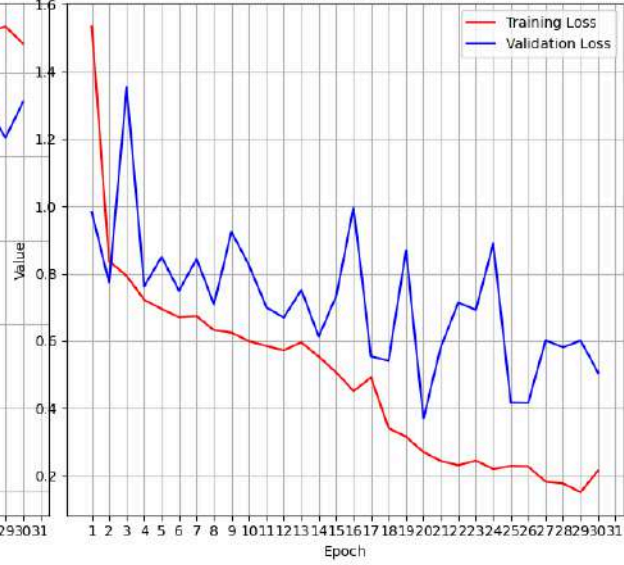


## SYSTEMS WITH MACHINE LEARNING - Task 4

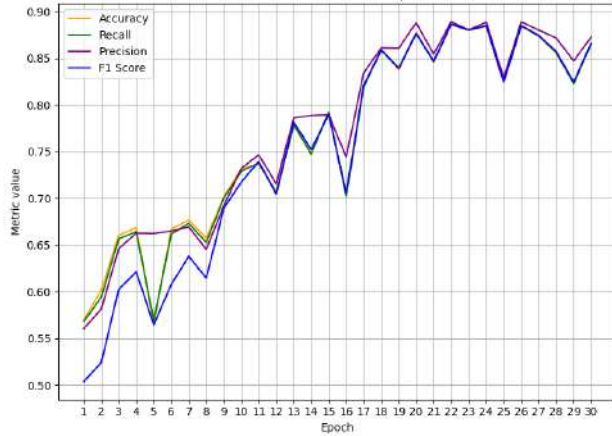
Training and Validation Accuracy



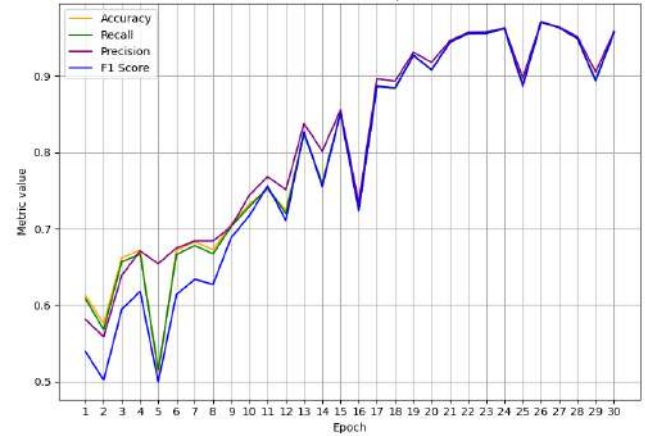
Training and Validation Loss



Validation Metrics over Epochs

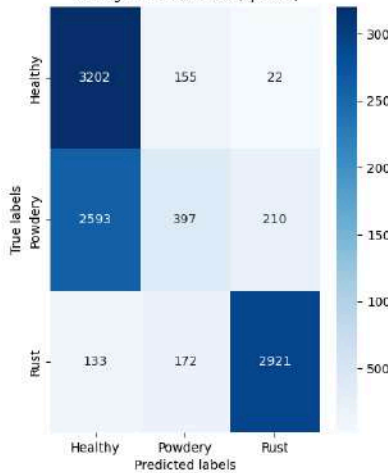


Train Metrics over Epochs

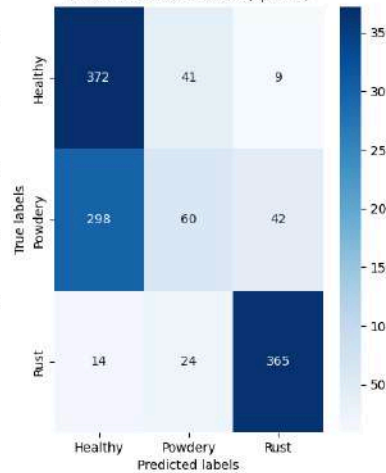


Confusion Matrices (Epoch 5)

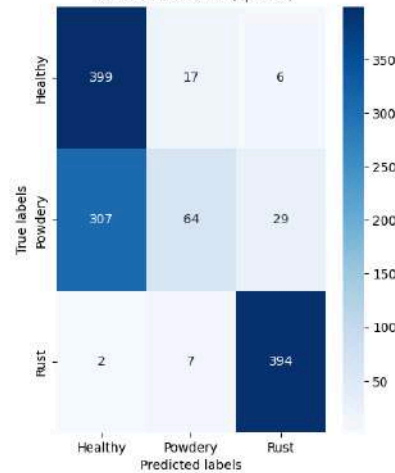
Training Confusion Matrix (Epoch 5)



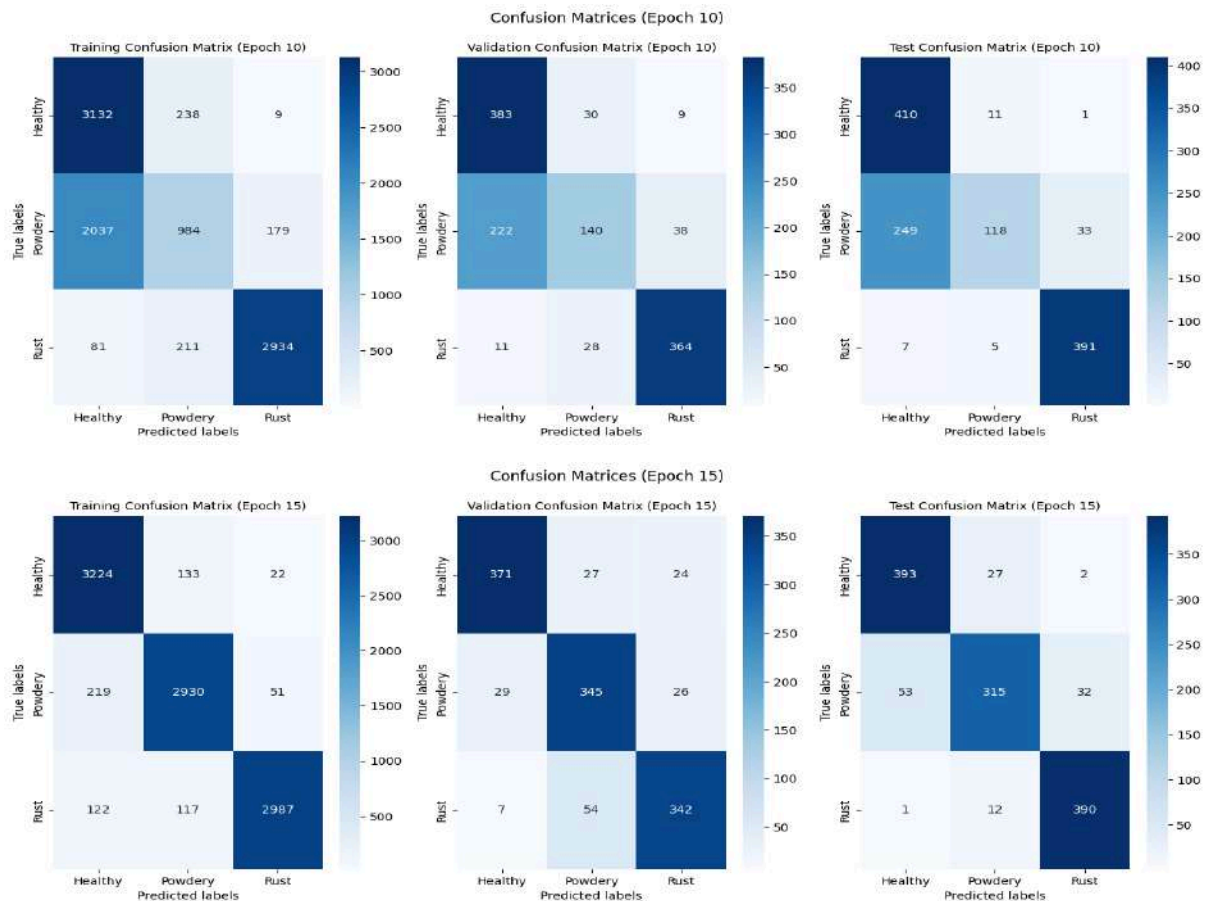
Validation Confusion Matrix (Epoch 5)



Test Confusion Matrix (Epoch 5)

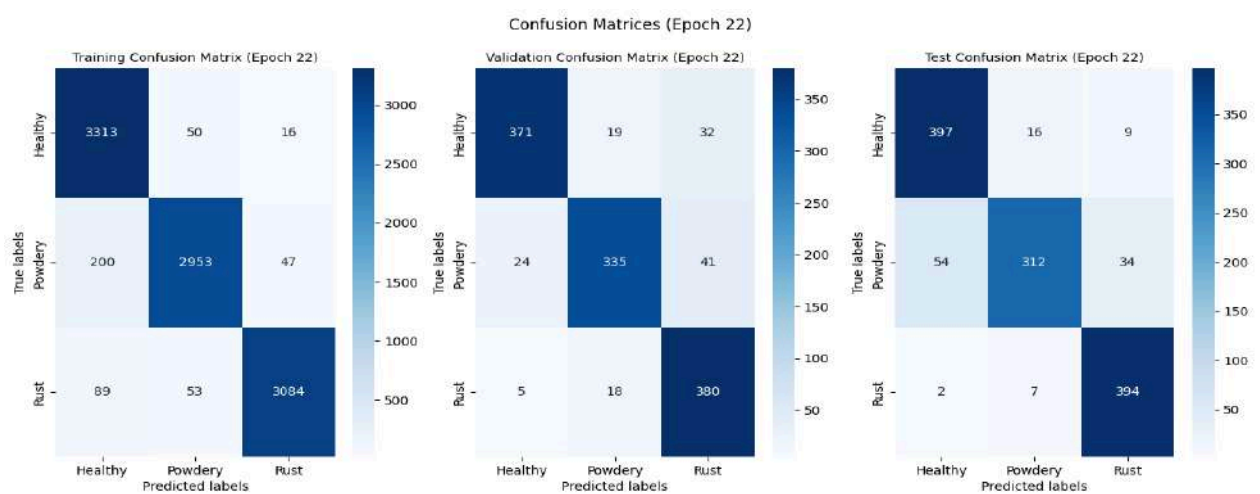


## SYSTEMS WITH MACHINE LEARNING - Task 4



The best model for every metric was from the 22nd epoch. Metrics from TEST set:

- accuracy: 0.9004081632653061
- recall: 0.8994752625451296
- precision: 0.9031082683983108
- f1: 0.8981678004535146



### **3.1.1. Observations**

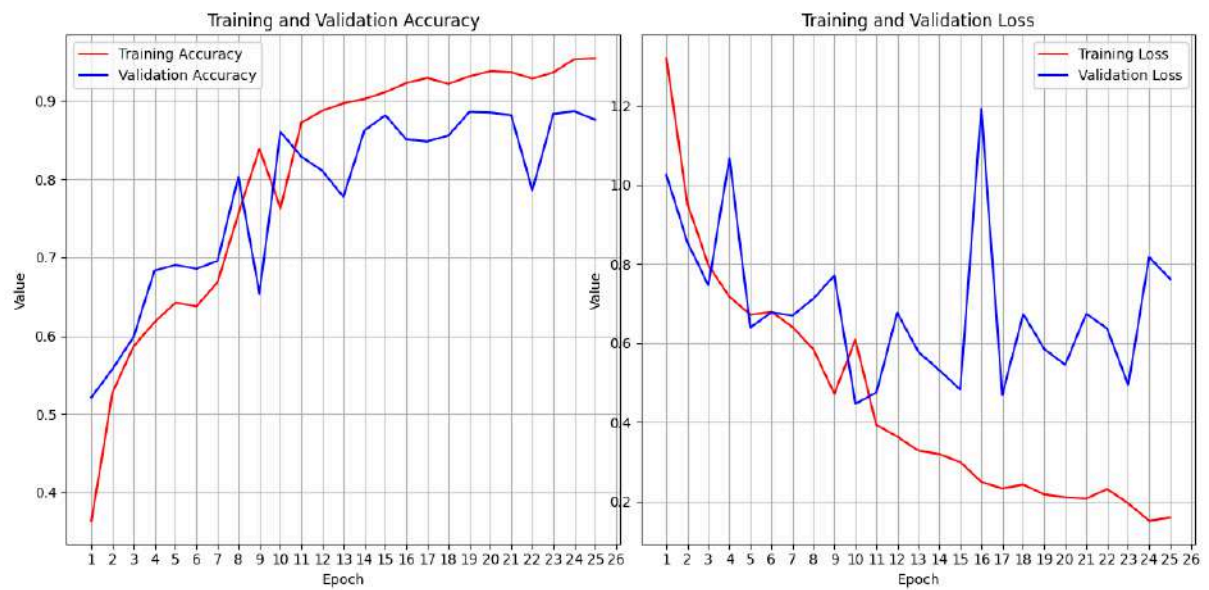
Metrics calculated for the training set have less deviation than those from the validation set. Increasing epochs increased the model's metrics. We can observe that the model's accuracy gain started to decline, which could hint that the model has started overfitting.

## **3.2. Grid Search (Batch size)**

For grid search we used batch size hyperparameter of the neural network model. We trained our model on batch sizes of 8, 16 (default), 32.

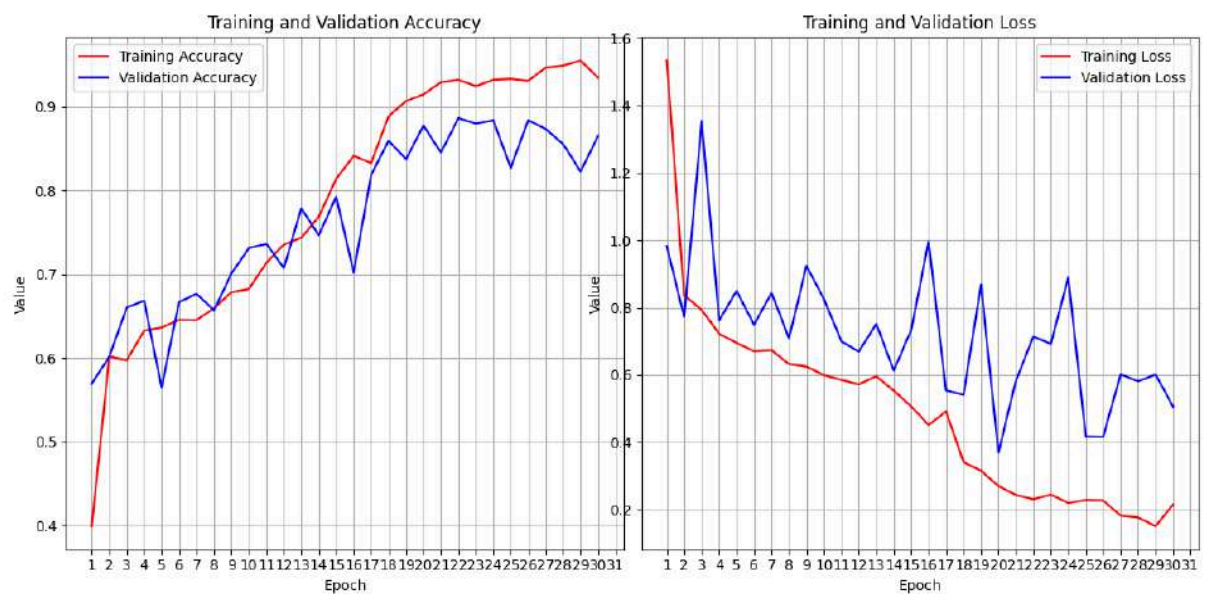
**Batch size = 8**

## SYSTEMS WITH MACHINE LEARNING - Task 4



Best : 10th epoch

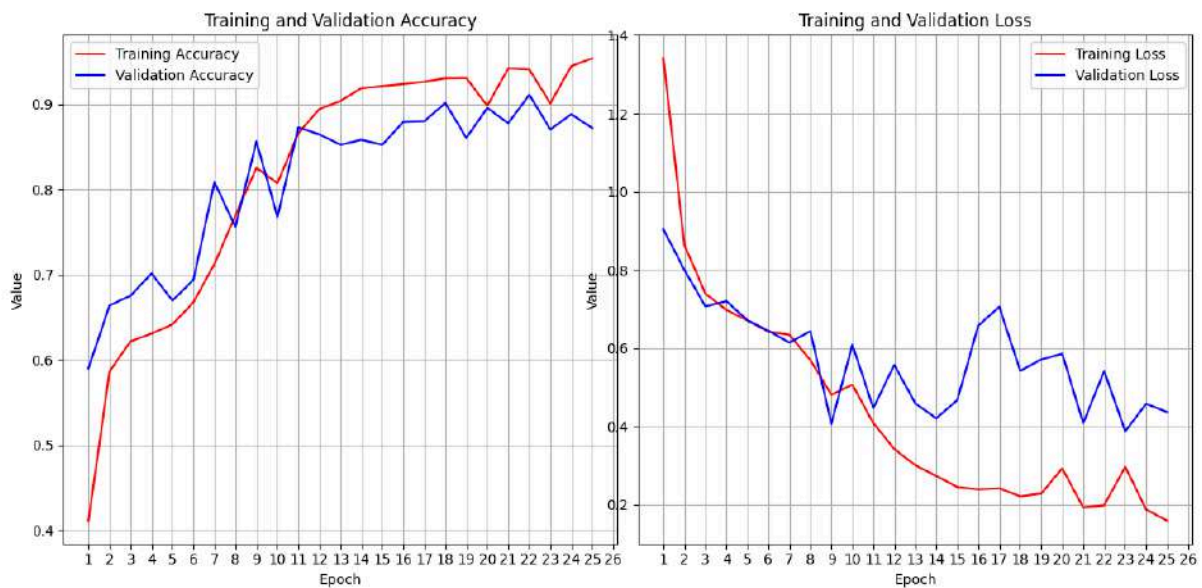
**Batch size = 16**



Best : 22nd epoch

**Batch size = 32**

## SYSTEMS WITH MACHINE LEARNING - Task 4



Best : 23rd epoch

Metrics for best models from each batch size:

Train	Accuracy	Recall	Precision	F1
<b>Batchsize8</b>	0.909739928607853 1	0.909420968495712 9	0.912668211570241 5	0.910018793737188 4
<b>Batchsize16</b>	0.970423253442121 3	0.970164365271875 8	0.970772536664343 9	0.970377593229530 4
<b>Batchsize32</b>	0.962468128505864 4	0.962686292377503 2	0.963708821754332 5	0.962596725542248 6

Validation	Accuracy	Recall	Precision	F1
<b>Batchsize8</b>	0.860408163265306 1	0.861025023618281 6	0.864290451934450 2	0.861100691573464 7
<b>Batchsize16</b>	0.886530612244897 9	0.886524986377837 7	0.888963243846281 7	0.886134171328292 5
<b>Batchsize32</b>	0.870204081632653 1	0.871142928823711 6	0.880948927985880 3	0.871534483562359 2

Test	Accuracy	Recall	Precision	F1
------	----------	--------	-----------	----

## SYSTEMS WITH MACHINE LEARNING - Task 4

<b>Batchsize8</b>	0.886530612244897 9	0.886133971516940 4	0.886151305259899 2	0.886100692253406 5
<b>Batchsize16</b>	0.900408163265306 1	0.899475262545129 6	0.903108268398310 8	0.898167800453514 6
<b>Batchsize32</b>	0.905306122448979 6	0.905592377861144 1	0.905382996008327 3	0.904841766076682 3

For metrics in Train and Validation sets, model with batch size of 16 had the best results. However in the Test set best results had model with batch size of 32. The worst results in all of sets had the model with batch size of 8.

In the differences between 16 and 32 batch sized models in test set were:

**Accuracy = 0.004897959183673417**

**Recall = 0.006117115316014554**

**Precision = 0.002274727610016458**

**F1 = 0.006673965623167688**

## 4. Other Possible Improvements

To improve the model one could try GridSearch or RandomSearch for better learning rates, different optimizers.

# Report 3

## Plant Disease Classification

Authors:

4. Krzysztof Nazar, 184698
5. Hubert Nacmer, 184546
6. Łukasz Kawa, 184948

GitHub: [link](#)

### 5. Basics

#### 5.1. Type of problem

Classification - classify the picture of a leaf into one of three predefined classes - Healthy, Rust or Powdery.

#### 5.2. Training methods

The training method we are going to use is supervised learning. For training we will use the image and its class.

#### 5.3. Machine Learning methods and models

Since our problem is in the field of image classification, Convolutional Neural Network is our choice for the ML model. Based on our research, we claim that these neural networks harness the power of Linear Algebra, specifically through convolution operations, to identify patterns within images. They consist of convolutional layers followed by pooling layers, which extract features and reduce dimensionality, respectively.

#### 5.4. Loss/fitness functions

Firstly, we were also thinking of using Weighted Crossentropy. However, we found out that it is useful in cases when the number of images in different classes is extremely

different. In our project the number of images in each of the given classes is very similar, therefore this type of loss function would not be a good choice.

Eventually, we are going to use **Categorical Crossentropy** as our loss function. From the Internet resources, we learnt that this is a standard choice for multi-class classification problems. It calculates the cross-entropy loss between the true labels and the predicted probabilities.

## 6. Training on SPLIT1 datasets - Krzysztof Nazar

### 6.1. Train and test

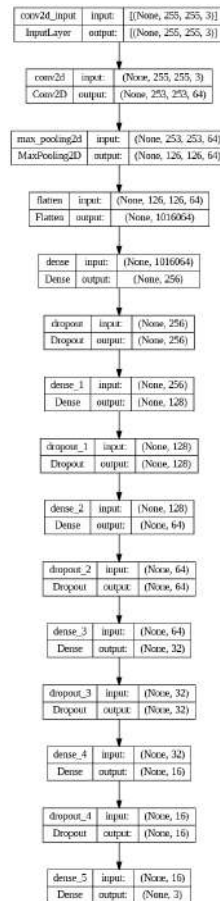
First of all, the training was performed on training data from SPLIT1 dataset - 1225 images in total. The validation set from SPLIT1 was used as the validation dataset - 153 images in total.

### 6.2. Training and testing analysis

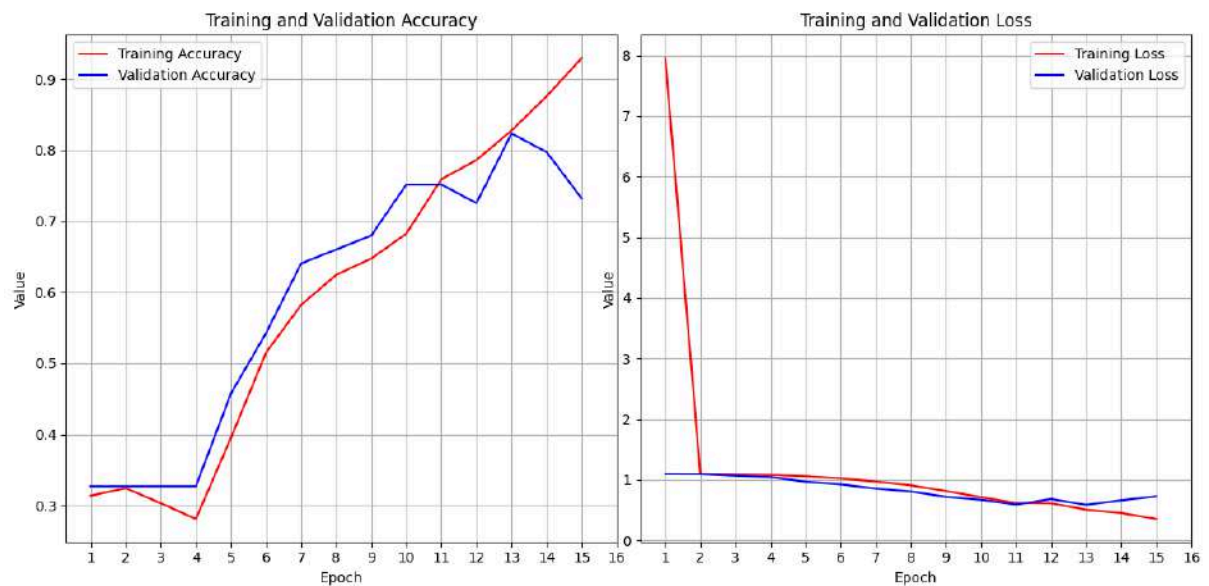
During training the following model was used:



## SYSTEMS WITH MACHINE LEARNING - Task 4



The values of loss and accuracy both on training and validation sets are presented on the graphs below. These graphs present the changes in loss and accuracy during the training.

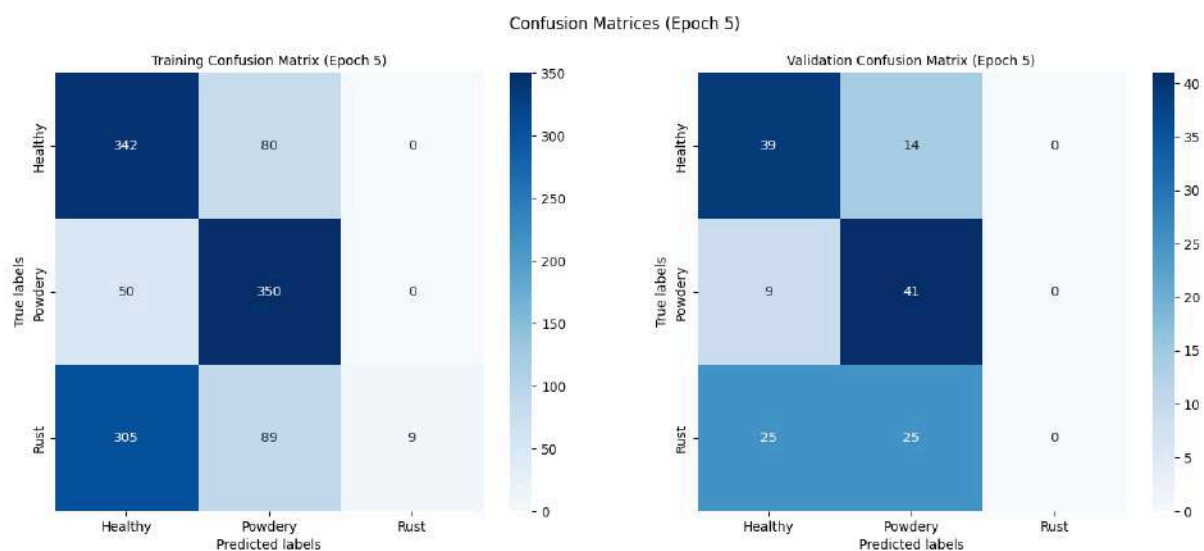


## SYSTEMS WITH MACHINE LEARNING - Task 4

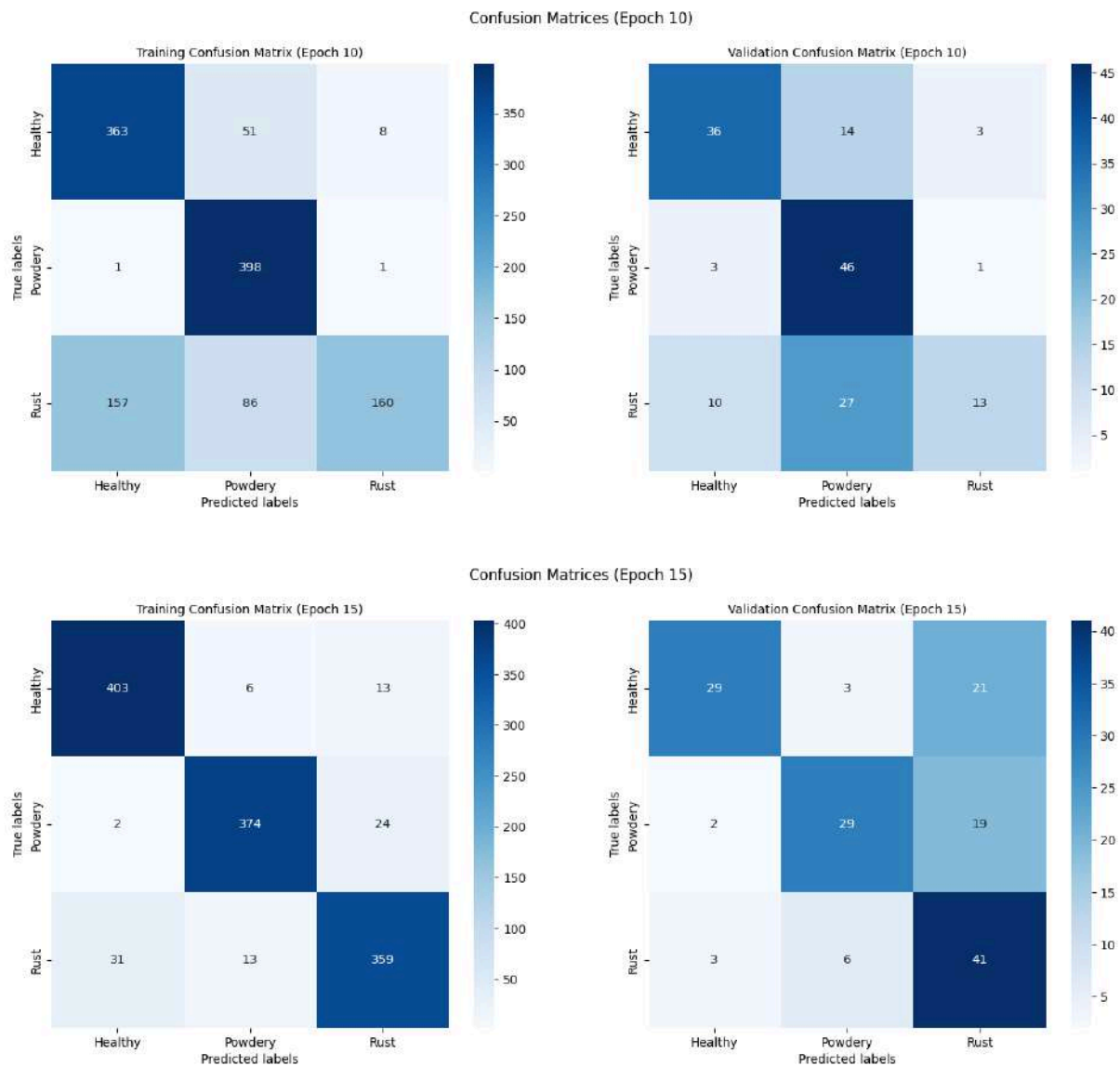
The first graph presents recorded values of accuracy of the model on the training and validation sets. The training accuracy increases in a stable manner. In the first epoch its value equals around 30%, however, later it increases at a fast pace reaching more than 90% in the last epoch. Interestingly, the validation accuracy remains almost constant at the beginning of training. After the 4th epoch it starts to increase gradually and at epoch 13<sup>th</sup> it reaches a maximum of about 82%. After this epoch, the validation accuracy starts to decrease to around 75% at epoch 15<sup>th</sup>.

The loss curves presented on the second graph present a significant decrease in the value of loss throughout the learning process of the model. At the beginning the training loss is much greater than the validation loss. Training loss drops quickly and then decreases at a similar pace as the validation loss. Eventually, both of them reach a value around 0.6 at the end of the training process.

Next, to better understand the training process, a confusion matrix for each 5<sup>th</sup> epoch was plotted and the results are presented below. The number of epochs can be seen in the main title of the graphics.

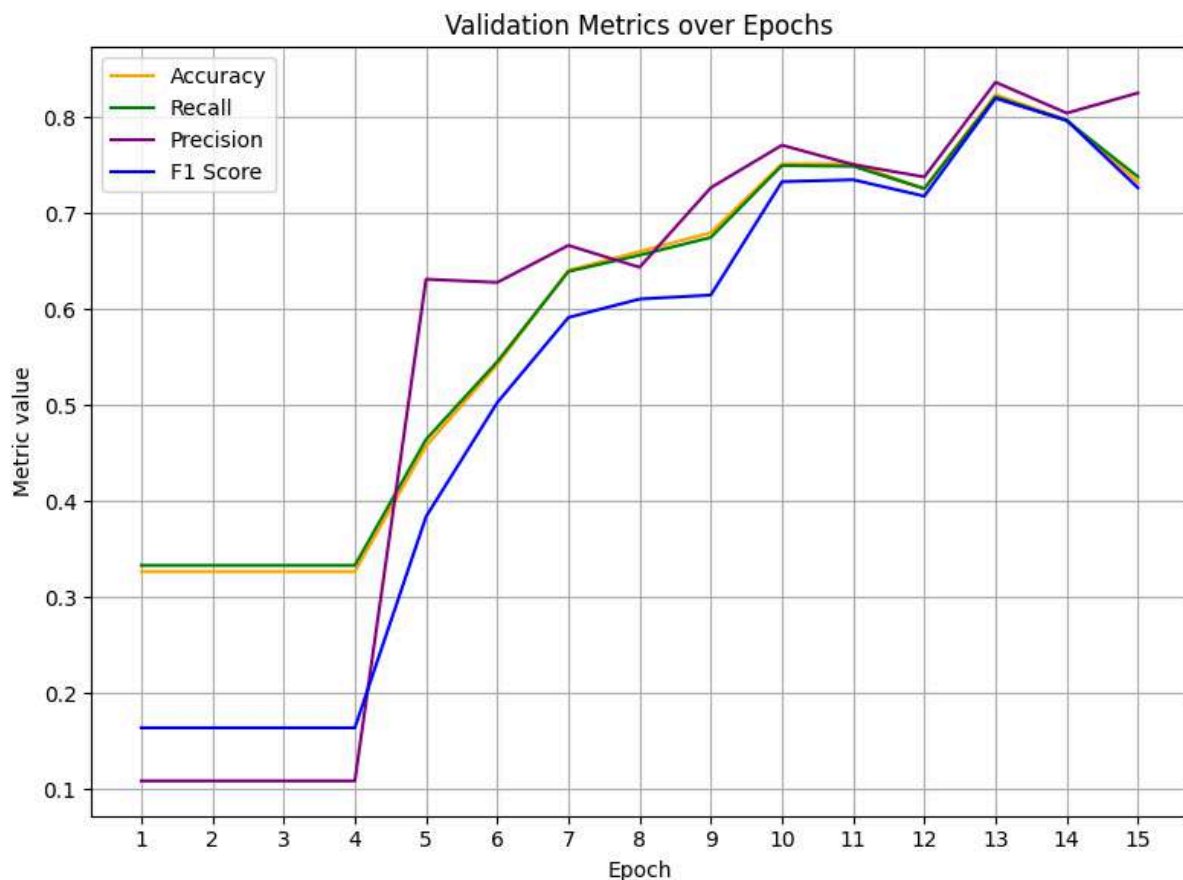


## SYSTEMS WITH MACHINE LEARNING - Task 4



Presented confusion matrices suggest that the model is better in classifying both seen and unseen images. The matrices which present the predictions on the training and validation set present relatively satisfactory results. For the training set, the model generally performs well, with high counts along the diagonal, indicating correct predictions. However, there are instances of confusion between classes, especially evident in class 2. Validation confusion matrices show good learning patterns.

### 6.3. Choosing the best model



In order to calculate additional metrics, the numbers of true positives, true negatives, false positives and false negatives were calculated. The values of calculated metrics remain almost constant at the beginning of the learning process, which seems uncommon, but can occur sometimes. After the 4<sup>th</sup> epoch the values of the metrics start to gradually increase. In the last part of the learning process, each of the metrics have a value of around 0.8, which is very satisfactory. These observations indicate that the model was learning well.

In order to get better results, more training data could be used. Adding some normalisation as well as augmentation would be probably profitable and enhance the learning process, hence the metrics would reach greater values. I believe that using more data would lead to more reasonable training and testing of the model.

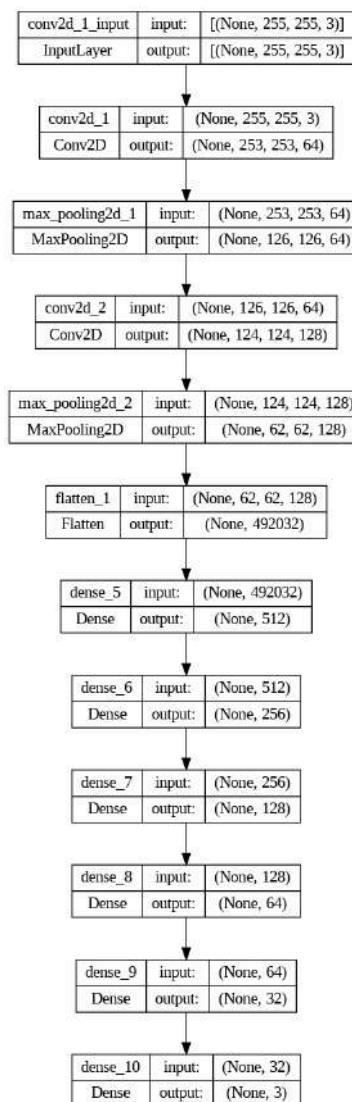
Based on the calculated metrics as well as the presented confusion matrices, the best model is the one from epoch 13<sup>th</sup>. The validation results suggest that it can predict unseen data with relatively high accuracy (around 80%). Other metrics - precision, recall

## SYSTEMS WITH MACHINE LEARNING - Task 4

and F1 score, were also relatively high reaching around 0.8 on the validation set. This model was saved as [MODEL1.h5](#).

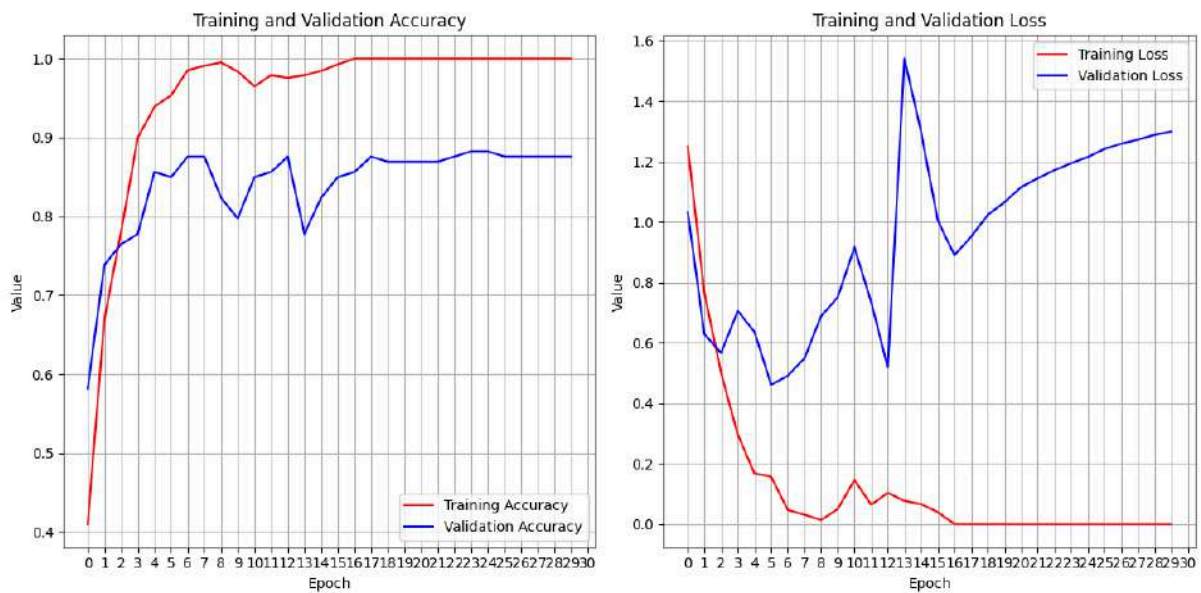
### 6.4. Model overfitting

In order to overfit the model, its architecture was slightly changed and more training epochs were used - 30 instead of 15 epochs. Now a deeper and wider model was used. It has several convolutional and dense layers, with a large number of neurons in each dense layer. There are no dropout layers. The batch size and the learning rate remained the same - 16 and 0.001 respectively. This is the architecture of the model:



The training and validating process is presented below.

## SYSTEMS WITH MACHINE LEARNING - Task 4



The first graph indicates that the final accuracy of the model is close to 100%. It is gradually and steadily increasing over the training method. The loss value on training is generally decreasing, however at the 14<sup>th</sup> epoch it surprisingly increases its value. Then it starts to decrease again.

The second graph indicates that the validation loss initially decreases, indicating that the model is improving its performance on unseen data. However, after a certain point (around epoch 13), the validation loss starts to increase slightly and in the 18<sup>th</sup> epoch it reaches its maximum. The validation accuracy increases initially, but it seems to fluctuate after the 12<sup>th</sup> epoch, indicating that the model might be overfitting to the training data.

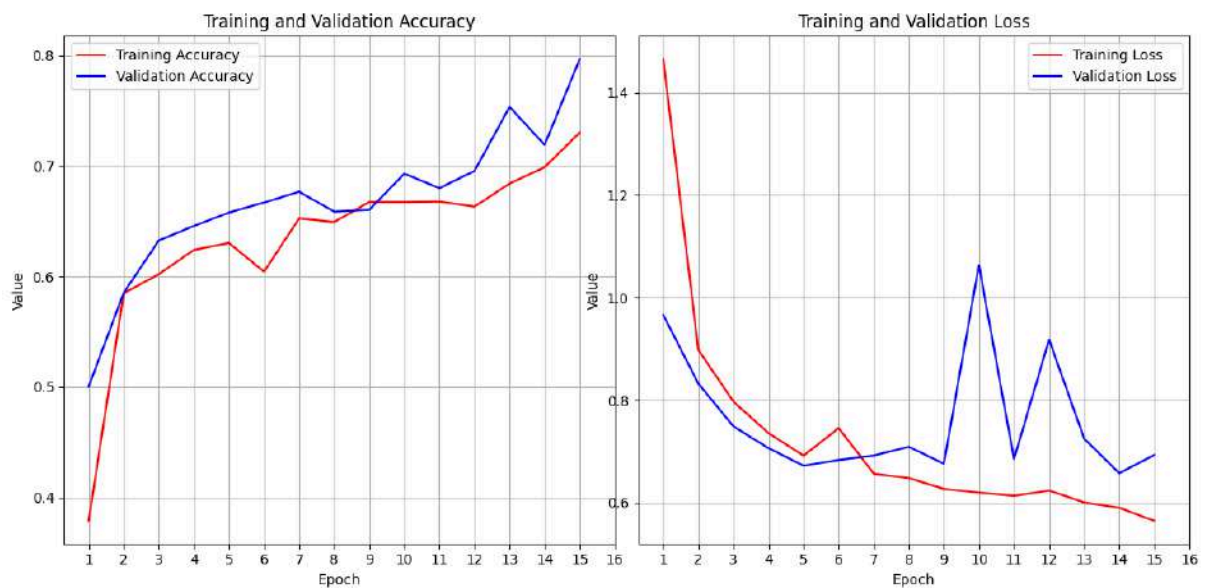
There's a noticeable gap between the training and validation accuracies, indicating overfitting. This gap suggests that the model is learning to classify the training data well but might not generalise as effectively to unseen data (validation set). These two graphs suggest that the model's performance on the validation set is worsening even as the training loss continues to decrease.

Based on these observations, it's clear that the model is overfitting, especially considering the increasing gap between the training and validation performance metrics and the increasing validation loss.

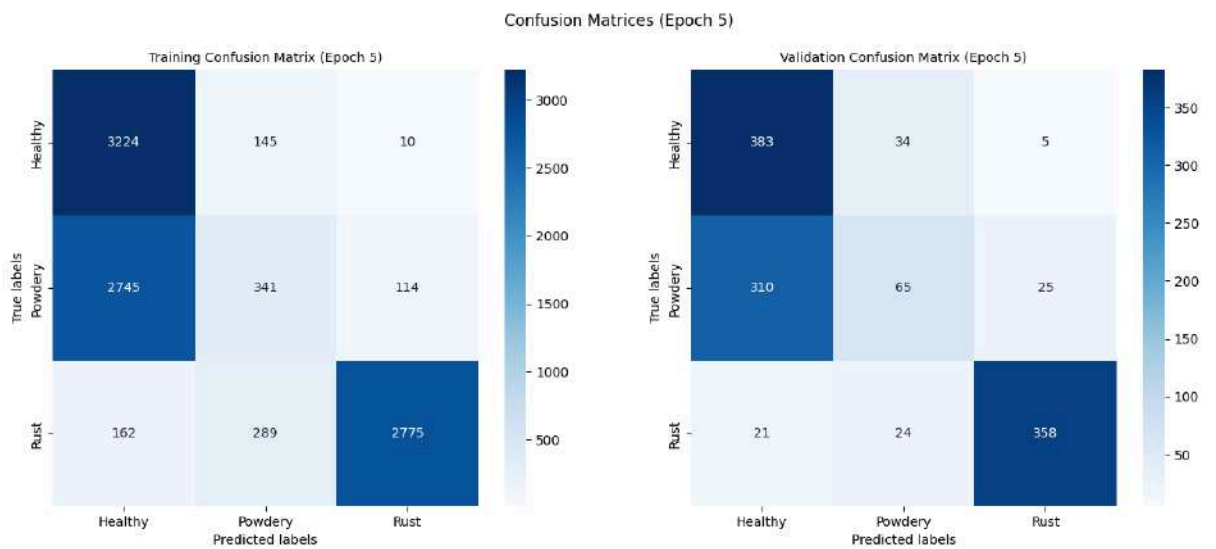
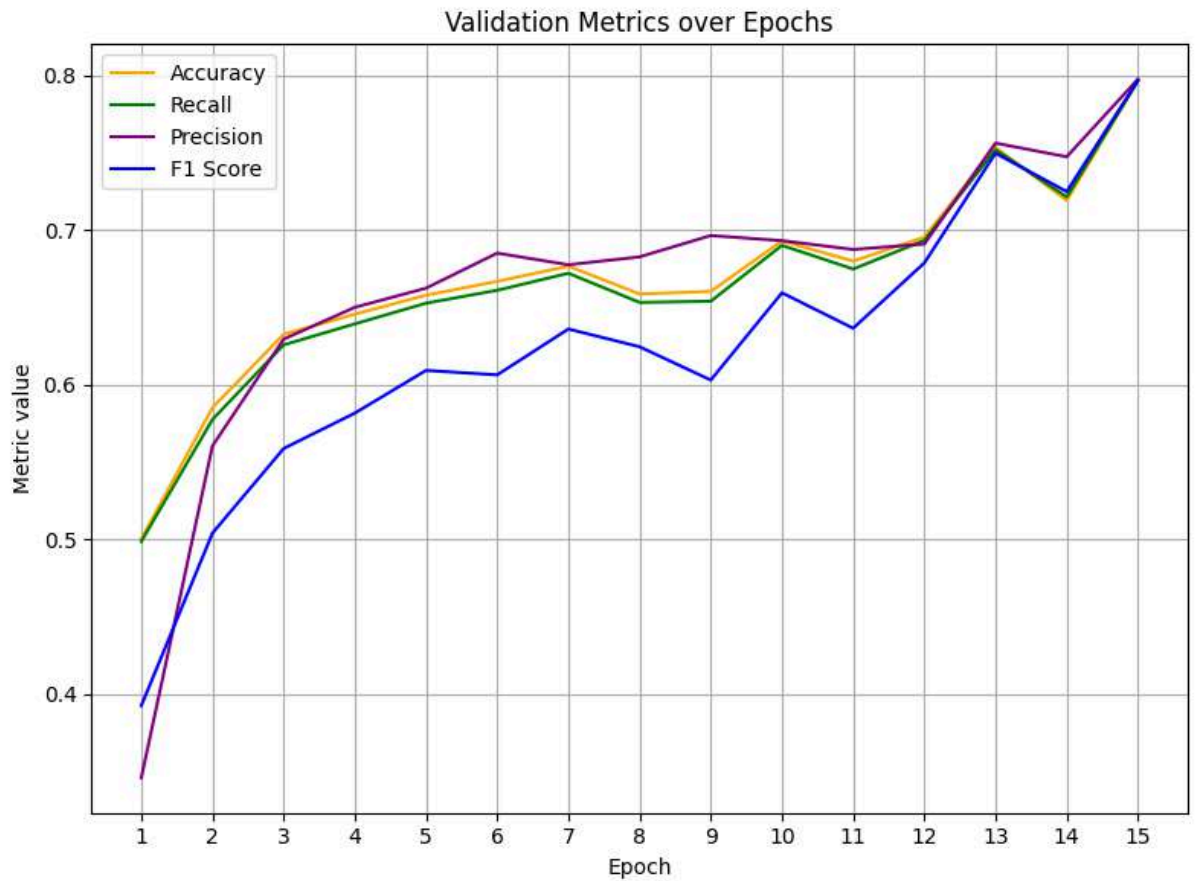
## 7. Training on SPLIT2 datasets - Łukasz Kawa

### 7.1. Train and test

The training was conducted on images whose size was adjusted by the `keras.preprocessing.image.ImageDataGenerator` function applying uniform scaling to all data. There are approximately 12 255 samples in two sets divided into 3 subsets



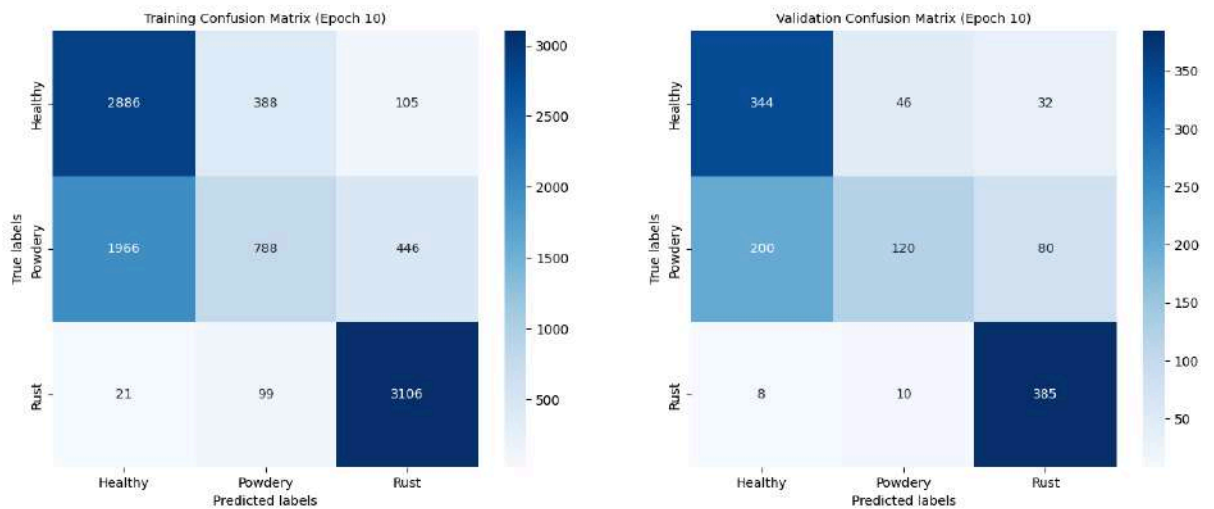
## SYSTEMS WITH MACHINE LEARNING - Task 4



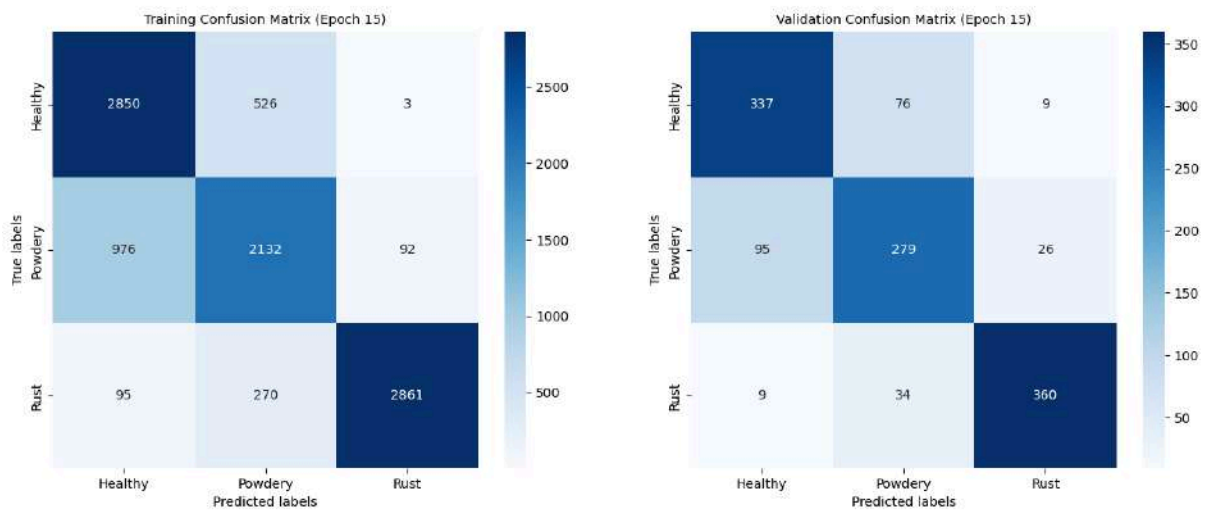


## SYSTEMS WITH MACHINE LEARNING - Task 4

Confusion Matrices (Epoch 10)



Confusion Matrices (Epoch 15)



### 7.2. Compare results to training on SPLIT1

	Split 1	Split 2
Number of data to process	Train: 1225 Validation: 153	Train: 9 805 Validation: 1 225
Average processing	~15 seconds	~120 seconds

## SYSTEMS WITH MACHINE LEARNING - Task 4

time for 1 epoch				
	Training	Validation	Training	Validation
Accuracy	0.958	0.824	0.803	0.797
Recall	0.958	0.821	0.802	0.796
Precision	0.960	0.836	0.811	0.798
F1 Score	0.957	0.820	0.804	0.797

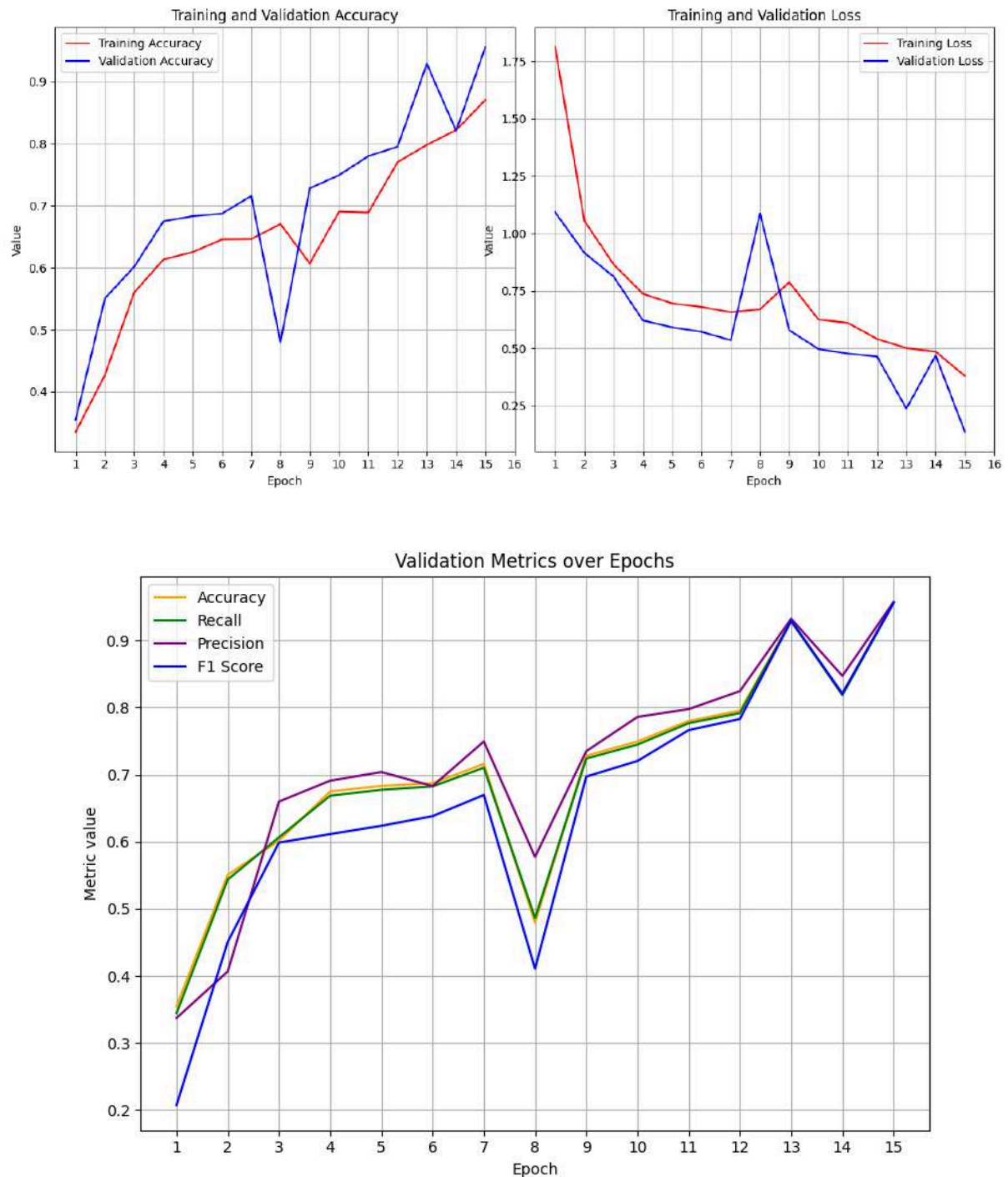
### 7.3. Choosing the best model

The best model was the 15th epoch checkpoint model.

[https://drive.google.com/file/d/1AhyHlbhgpN\\_Th68mgUMpnruQDUye3iAD/view?usp=drive\\_link](https://drive.google.com/file/d/1AhyHlbhgpN_Th68mgUMpnruQDUye3iAD/view?usp=drive_link)

## 8. Training on SPLIT3 datasets

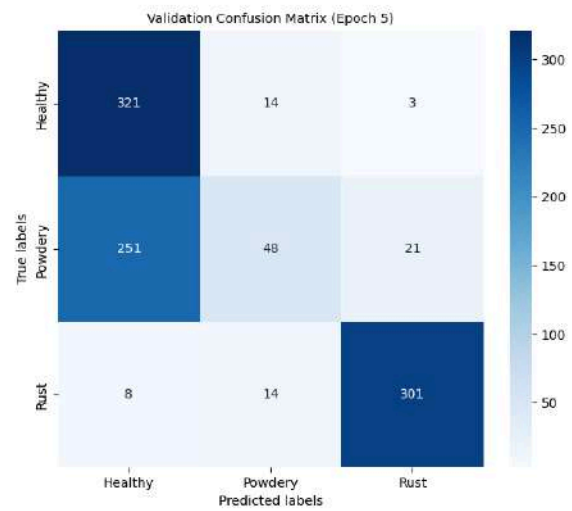
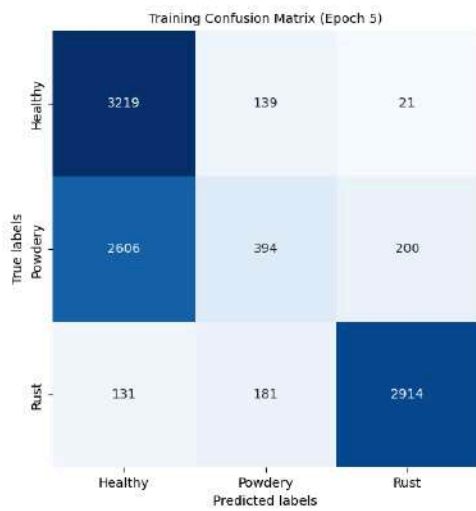
### 8.1. Train and test



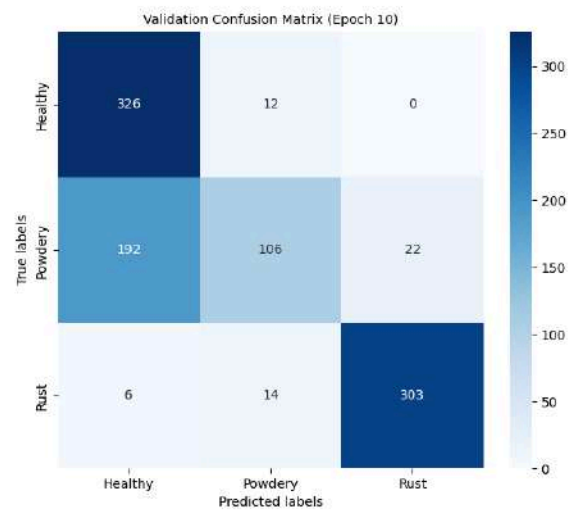
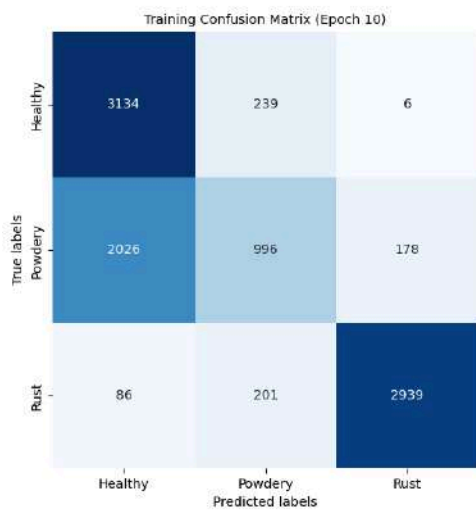
Loss: 0.3794 - accuracy: 0.8706 - val\_loss: 0.1365 - val\_accuracy: 0.9562

## SYSTEMS WITH MACHINE LEARNING - Task 4

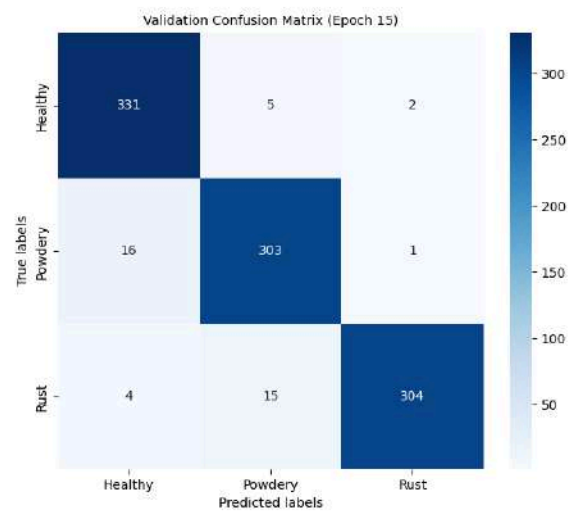
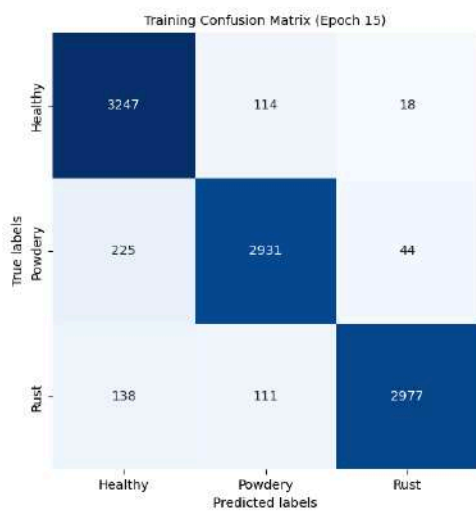
Confusion Matrices (Epoch 5)



Confusion Matrices (Epoch 10)



Confusion Matrices (Epoch 15)



## 8.2. Compare results to training on SPLIT 2:

	Split 2		Split 3	
Number of data to process	Train : 9 805 Val : 1 225		Train : 9 805 Val : 981	
Average processing time for 1 epoch	~2 min		~2 min	
	Training	Validation	Training	Validation
Accuracy	0.803	0.797	0.933	0.956
Recall	0.802	0.796	0.933	0.956
Precision	0.811	0.798	0.935	0.957
F1 Score	0.804	0.797	0.934	0.956

## 8.3. Choosing the best model

The best model was the model from the 15th epoch.

[https://drive.google.com/file/d/1eeJISA01FFswWUCwHwZTU99CpUOWXl4j/view?usp=drive\\_link](https://drive.google.com/file/d/1eeJISA01FFswWUCwHwZTU99CpUOWXl4j/view?usp=drive_link)

NOTE. SPLIT3 shall be exactly the same as SPLIT2, with one exception: VAL set must be added to TRAIN set.

## Report 2

# Plant Disease Classification

Authors:

7. Krzysztof Nazar, 184698
8. Hubert Nacmer, 184546
9. Łukasz Kawa, 184948

GitHub: [link](#)

## 9. Data description

All we need is the "Plant disease recognition" dataset, which is available online on Kaggle using [this link](#). This dataset contains images with three labels, "Healthy", "Powdery", "Rust" referring to plant conditions. There are 1532 images in the whole dataset. Originally they are divided into train, test, and validation sets. These images were scrapped from online sources. Every image is in a JPG format. The images have different sizes - this will be handled in the later steps. The whole dataset weighs 1.35GB. Each image weighs around 945 kB and the average image size is 3982x2700 pixels - such quality is detailed enough to train, test and validate the ML model accurately. Images are in colour. Each image is classified into one of the three possible classes. Images are not blurred and are of reasonably high quality. The final tests will be conducted on a test set of images appropriately selected from the whole dataset.

## 10. Dataset content

Whole dataset contains 1532 of single RBG images, which are divided into 3 categories:

- Healthy - 528 images
- Powdery - 500 images
- Rust - 504 images

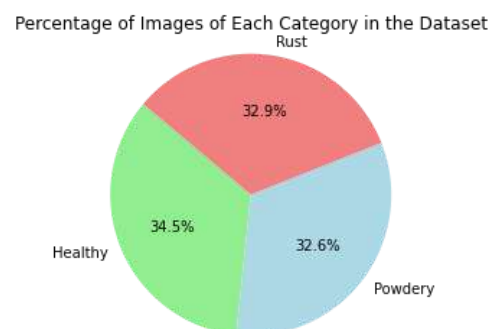
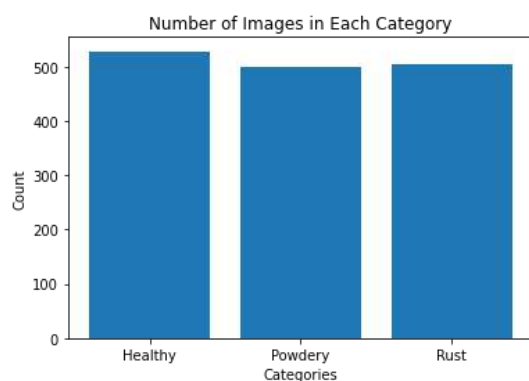
The picture below presents a sample of randomly selected images from the original dataset. Each image has an assigned label created by the creators of the sourced dataset. A single image is assigned to only one of the three possible categories.

## SYSTEMS WITH MACHINE LEARNING - Task 4

Labelling was performed in a boolean form. This means that each image is selected to just one category - there is no information about the extent of how healthy the leaf presented on the image is.



The charts below illustrate the distribution of each category within the given dataset. There is a balanced distribution between the categories in the dataset, with almost an equal number of images across all classes. Such a good balance in class distribution suggests that our solution will be equipped to effectively classify all categories without bias towards any particular class. What is more, the balanced distribution lowers the risk of encountering challenges in correctly classifying rarer categories, ensuring a fair and accurate classification process.



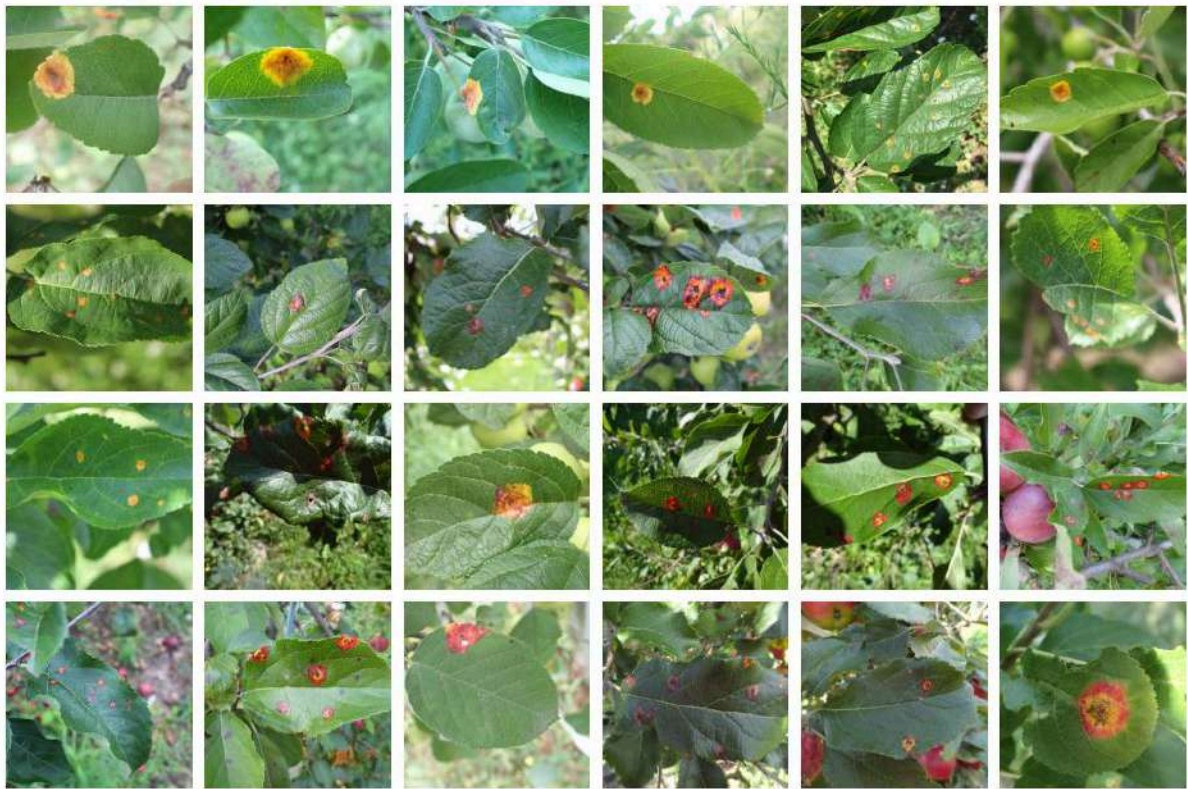


## 11. Data relations

Each image predominantly features lush greenery, with leaves dominating the scene. However, some plants bear fruits, potentially complicating classification in our learning model.

The "Rusty" category stands out as the most distinctive. It exhibits characteristic warm-coloured dots, which could prove invaluable for our classification model. While many images with warm hues are correctly identified as rusty plants, some containing tree fruits may be misclassified.

### Labelled as Rusty



The "Powdery" class is characterised predominantly by shades of grey. To determine if a plant is powdery, it's often necessary to observe it from below. Occasionally, about 10% of the images of powdery plants may depict a human hand holding the plant in a certain position. This poses a challenge, as such images may be misidentified as rusty plants due to the presence of warm skin tones.

### Labelled as Powdery



## SYSTEMS WITH MACHINE LEARNING - Task 4



The "Healthy" category is defined by an abundance of vibrant green hues, with minimal variation among images. However, some photos may exhibit reflections from the sun, which could potentially impact the classification outcome.

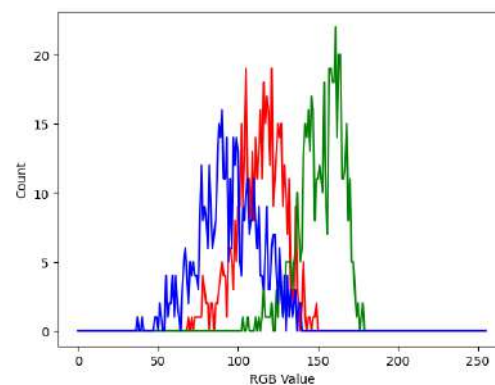
### **Labelled as Healthy**

## SYSTEMS WITH MACHINE LEARNING - Task 4



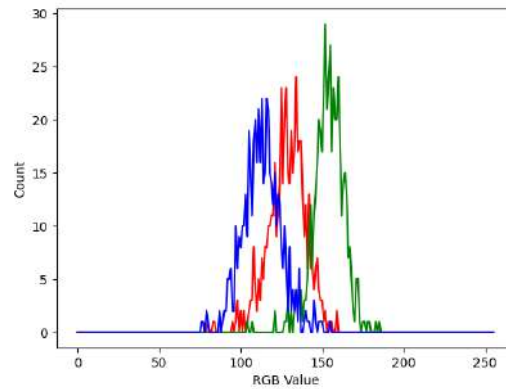
To verify whether a specific class can contain elements correlated with this, we've created diagrams representing the mean RGB values for each image within each class. Upon analysis, we observe that in the healthy class, the values are more dispersed compared to the powdery class, albeit still within a certain range, suggesting potential differences in grayscale. In the rust class, we note even greater dispersion in values compared to the other classes, indicating the presence of more varied colours in the images. This could imply that the plants in these images exhibit colours other than green, possibly indicating the presence of red dots, indicative of an unhealthy plant.

### Healthy class

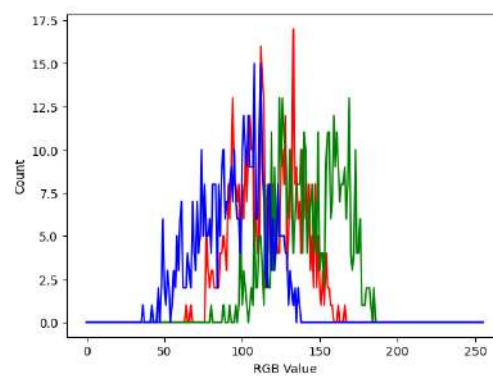


### Powdery class

## SYSTEMS WITH MACHINE LEARNING - Task 4



**Rust class**



## 12. Errors and noise



Most of the dataset images are of high quality, however some of the images in dataset, about 0.5%, are out of focus or blurry.

Backgrounds of images are similar but diverse, which could help with focusing on leaves and not on the background.





## SYSTEMS WITH MACHINE LEARNING - Task 4

Lighting is also different on some of the images. Some of them are in bright light and others are in dimmer light or in shadows.

We found only one incorrectly labelled image as healthy.



### 13. Data difficulty

Most of the images are clear and we can clearly assess to which class each leaf should be assigned.

Some of the images have backgrounds that could disrupt classification. E.g. leaves with red apples in the background, which could be interpreted as rusty leaves or rusty leaves with a few dots that could be classified as healthy, because it's similar to a few apples in the background.

Some of the images are also blurry as mentioned above. However since the blur is not very impactful, we do not consider it a big issue.

No special weighting during training, it's unnecessary for our case, every class is equally important.

### 14. Data representation

For the data representation we chose raw image data format. By default, authors of the dataset stored the images in training, testing and validation splits, and further split the data by directories named after classes. We merged the 3 main splits, so that the data was only split by the labels. Further we will split the data ourselves into training, testing and validation sets.

We will use the path of the images, mainly the name of the directory, as a label for classification.

## 15. Data normalisation

Since the dataset consists of images with different resolutions we had to make it so the images had the same resolution. To achieve this we used library [scikit-image](#) for resizing each image to a mean resolution of all the images (3982x2700px). The second method we used was to crop all of the images to the smallest width and smallest height of the whole dataset (2421x1728 px).

Example of raw, cropped and resized image:



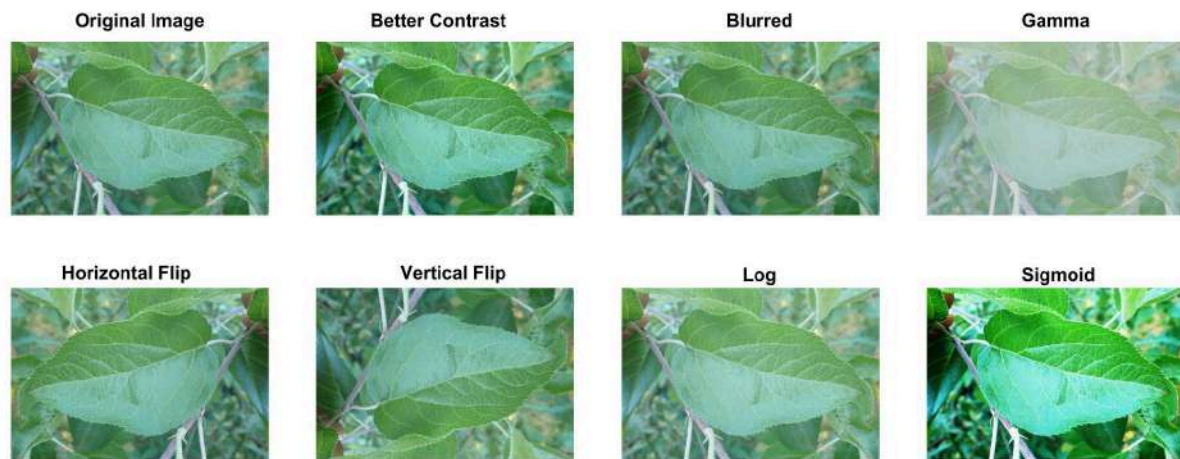
Another normalisation of the dataset is to change the type and range of image arrays from integers (0-255) to floating point numbers (0-1).

## 16. Data augmentation

The library [scikit-image](#) was used for data augmentation. The library is a collection of algorithms for image processing. The library was used to create several copies of each image. The transformations included:

- Rescale intensity (change contrast)
- Gamma correction
- Logarithmic correction
- Sigmoid correction
- Horizontal flip
- Vertical flip
- Blur image

An example of an augmented image is presented below. The original image is shown in the left upper corner.



## 17. Data splits

### 17.1. Split 1

*SPLIT 1. Original class distribution / proportion, original data (no normalisation, no augmentation)*

To achieve a split, we calculate the number of images each set should contain. For our example, we've established proportions for various classes in the project: 80% for training, 10% for testing, and 10% for validation.

```
# PROPORTIONS  
TRAIN = 0.8  
VAL = 0.1  
TEST = 0.1
```

Subsequently, we've organised folders for training, testing, and validation, each containing directories for every class in the dataset.

## SYSTEMS WITH MACHINE LEARNING - Task 4

```
train_dir = "train"
val_dir = "val"
test_dir = "test"
```

```
# split 1
for directory in images_directories:
    directory_path = f"{merged_data_dir}\\{directory}"
    files = os.listdir(directory_path)
    train_count = round(len(files) * TRAIN)
    val_count = round(len(files) * VAL)
    test_count = round(len(files) * TEST)
    for filename in files:
        if filename.endswith('.jpg') or filename.endswith('.jpeg') or filename.endswith('.png'):
            image_path = os.path.join(directory_path, filename)
            if train_count != 0:
                os.makedirs(os.path.dirname(f"{all_splits_dir}\\split1\\{train_dir}\\{directory}\\"), exist_ok=True)
                destination = f"{all_splits_dir}\\split1\\{train_dir}\\{directory}\\{filename}"
                shutil.copyfile(image_path, destination)
                train_count -= 1
            elif val_count != 0:
                os.makedirs(os.path.dirname(f"{all_splits_dir}\\split1\\{val_dir}\\{directory}\\"), exist_ok=True)
                destination = f"{all_splits_dir}\\split1\\{val_dir}\\{directory}\\{filename}"
                shutil.copyfile(image_path, destination)
                val_count -= 1
            elif test_count != 0:
                os.makedirs(os.path.dirname(f"{all_splits_dir}\\split1\\{test_dir}\\{directory}\\"), exist_ok=True)
                destination = f"{all_splits_dir}\\split1\\{test_dir}\\{directory}\\{filename}"
                shutil.copyfile(image_path, destination)
                test_count -= 1
```

## 17.2. Split 2

*SPLIT 2. Uniform class distribution / proportion, normalized and augmented data*

We utilize a resizing functionality for a set of images, followed by an augmentation process aimed at expanding our dataset for improved model learning.

```
# split 2 - resized -> augmented -> splited
for directory in images_directories:
    directory_path = f"{augmented_data_dir}\\{directory}"
    files = os.listdir(directory_path)
    train_count = round(len(files) * TRAIN)
    val_count = round(len(files) * VAL)
    test_count = round(len(files) * TEST)
    for filename in files:
        if filename.endswith('.jpg') or filename.endswith('.jpeg') or filename.endswith('.png'):
            image_path = os.path.join(directory_path, filename)
            if train_count != 0:
                os.makedirs(os.path.dirname(f"{all_splits_dir}\\split2\\{train_dir}\\{directory}\\{filename}"), exist_ok=True)
                destination = f"{all_splits_dir}\\split2\\{train_dir}\\{directory}\\{filename}"
                shutil.copyfile(image_path, destination)
                train_count -= 1
            elif val_count != 0:
                os.makedirs(os.path.dirname(f"{all_splits_dir}\\split2\\{val_dir}\\{directory}\\{filename}"), exist_ok=True)
                destination = f"{all_splits_dir}\\split2\\{val_dir}\\{directory}\\{filename}"
                shutil.copyfile(image_path, destination)
                val_count -= 1
            elif test_count != 0:
                os.makedirs(os.path.dirname(f"{all_splits_dir}\\split2\\{test_dir}\\{directory}\\{filename}"), exist_ok=True)
                destination = f"{all_splits_dir}\\split2\\{test_dir}\\{directory}\\{filename}"
                shutil.copyfile(image_path, destination)
                test_count -= 1
```

17.3.

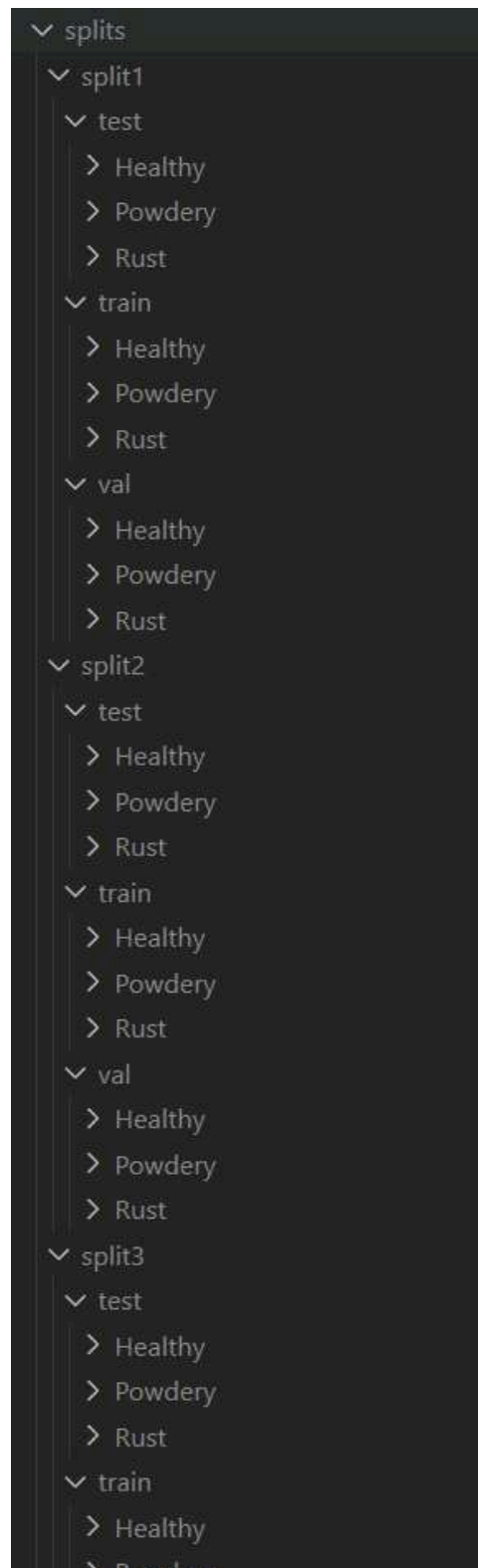
## 17.4. Split 3

*SPLIT 3. Like "SPLIT2", but instead of original VAL set use part of TRAIN set, keeping TRAIN and TEST the same as before*

In split 3, we create the validation set by selecting a subset of the training directory from split 2. The size of this subset is determined based on the computed validation proportions.



## SYSTEMS WITH MACHINE LEARNING - Task 4



# Report 1

## Plant Disease Classification

Authors:

- Krzysztof Nazar, 184698
- Hubert Nacmer, 184546
- Łukasz Kawa, 184948

### Problem Definition

The objective of the system is to accurately classify plant images into three categories: Healthy, Powdery, and Rust. Healthy plants exhibit no signs of disease, while powdery mildew and rust infections are characterised by distinct visual symptoms such as powdery white spots and rust-coloured spores, respectively. The classification of plant diseases is crucial for implementing timely interventions and mitigating crop losses, ultimately enhancing agricultural sustainability and food production.

The data used in this project comes from the Plant Disease Classification dataset available on Kaggle using this [link](#).

### System input

The dataset contains three labels: Healthy, Powdery, Rust referring to plant conditions. There are 1532 images of different sizes. The whole dataset weighs 1.35GB. Each image weighs around 945.6 kB and the average image size is 3982x2700 pixels. Images are in colour: RGB colorspace, 8-bit per colour. Each image is classified into one class. Images are not blurred and are of reasonably high quality.

An example input is presented below:

## SYSTEMS WITH MACHINE LEARNING - Task 4



### System output

Identify the specific class of plants based on its condition.

There are 3 possible classes:

1. Healthy

## SYSTEMS WITH MACHINE LEARNING - Task 4

2. Powdery

3. Rust



Plant is classified as **healthy** if there are no signs of illness.

**Powdery** mildew is a fungal disease that affects a wide range of plants.

**Rust** is a common fungal disease of many different plants in the flower garden

There is no planned post-processing of machine learning output. Raw confidence (probability) values for each class will be returned. Therefore, the system will have three numerical outputs, one for each class. Each value will be a real number in range [0,1], where 0 = 0% confidence, and 1 = 100% confidence.

Example output for input:

	Healthy: 0.92 Powdery: 0.06 Rust: 0.02
	Healthy: 0.05 Powdery: 0.94 Rust: 0.01

## SYSTEMS WITH MACHINE LEARNING - Task 4



Healthy: 0.02

Powdery: 0.01

Rust: 0.97