Project: Neural Networks – Leaf image classification

Course: Algorithms for Massive Data

Name: Xaver Brückner

MAT: 989819

Xaver.brueckner@studenti.unimi.it

June 2023

Inhalt

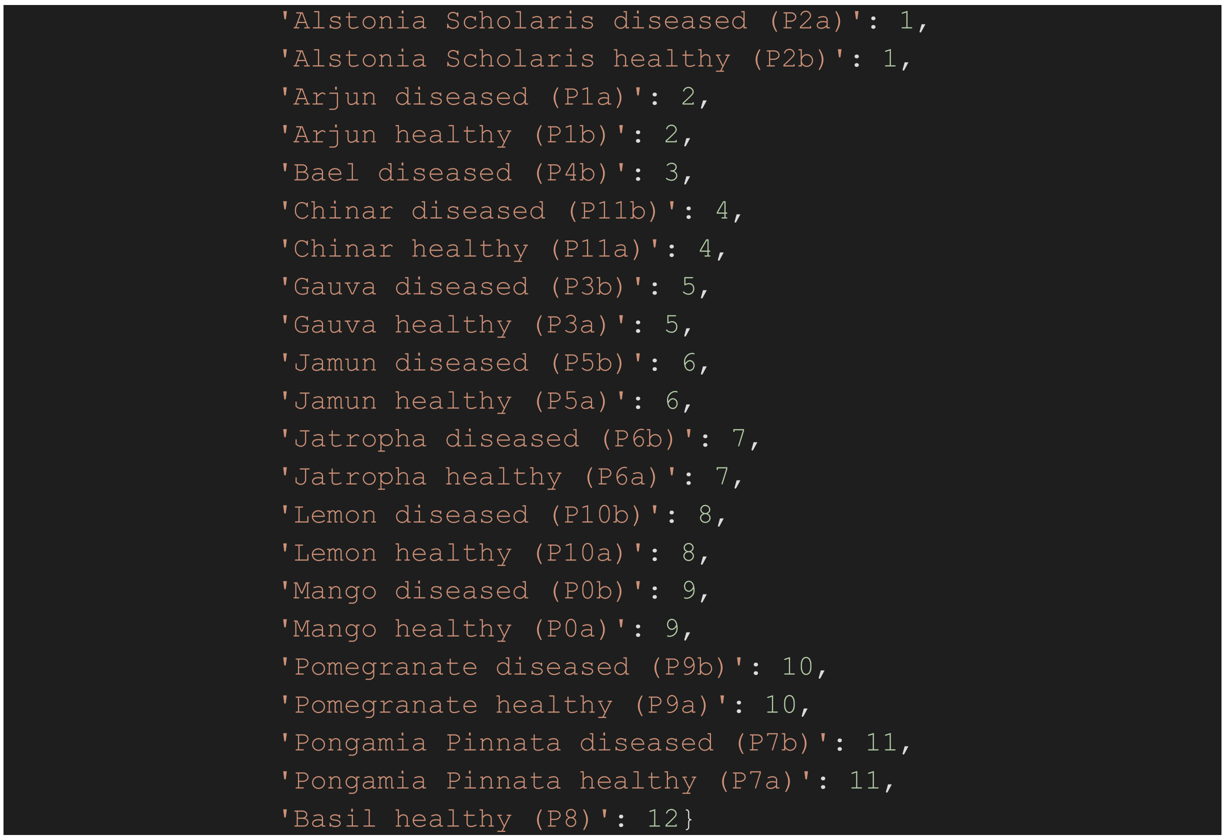
Abstract

This project explores the use of Convolutional Neural Networks (CNNs) to classify plant leaves based on their characteristics in a multiclass image classification problem. Performance evaluations reveal the CNN models provide good accuracy in classifying plant leaves with a 90.91% accuracy achieved in the best-performing one.

1. Dataset

The dataset considered in this report is the ‘Plant Leaves for Image Classification’ dataset from Kaggle. It consists of a total of 4503 images, which contain the leaves of 12 different plants, namely Mango, Arjun, Alstonia Scholaris, Guava, Bael, Jamun, Jatropha, Pongamia Pinnata, Basil, Pomegranate, Lemon, and Chinar in them.
There was also information on the leaf’s current health, which was either healthy or diseased included in the data. This aspect was discarded, and the focus of this report was centred around correctly classifying the species each leaf belonged to.

Therefore, each pair of leaf labels healthy & diseased were combined into a single class in the following manner.



```
'Alstonia Scholaris diseased (P2a)': 1,
'Alstonia Scholaris healthy (P2b)': 1,
'Arjun diseased (P1a)': 2,
'Arjun healthy (P1b)': 2,
'Bael diseased (P4b)': 3,
'Chinar diseased (P11b)': 4,
'Chinar healthy (P11a)': 4,
'Gauva diseased (P3b)': 5,
'Gauva healthy (P3a)': 5,
'Jamun diseased (P5b)': 6,
'Jamun healthy (P5a)': 6,
'Jatropha diseased (P6b)': 7,
'Jatropha healthy (P6a)': 7,
'Lemon diseased (P10b)': 8,
'Lemon healthy (P10a)': 8,
'Mango diseased (P0b)': 9,
'Mango healthy (P0a)': 9,
'Pomegranate diseased (P9b)': 10,
'Pomegranate healthy (P9a)': 10,
'Pongamia Pinnata diseased (P7b)': 11,
'Pongamia Pinnata healthy (P7a)': 11,
'Basil healthy (P8)': 12}
```

The data has also been split into train, validation, and test sets beforehand. The split is as follows:

- Training set	- 4274 images
- Validation set	- 110 images
- Test set	- 110 images

Then, the data was loaded in at runtime via a simple data loader. Another option would have been to use a data generator to provide the batches to the individual models, as shown at the end of the notebook. This solution would especially be superior in case of RAM shortage. In this case, the data loader was sufficient and guaranteed a fast training time for the models.

2. Exploratory data analysis

A short exploratory data analysis has been carried out to get an idea of the distribution of the data. In the first step, the number of occurrences of each plant species have been compared to find out if the dataset is imbalanced or balanced.

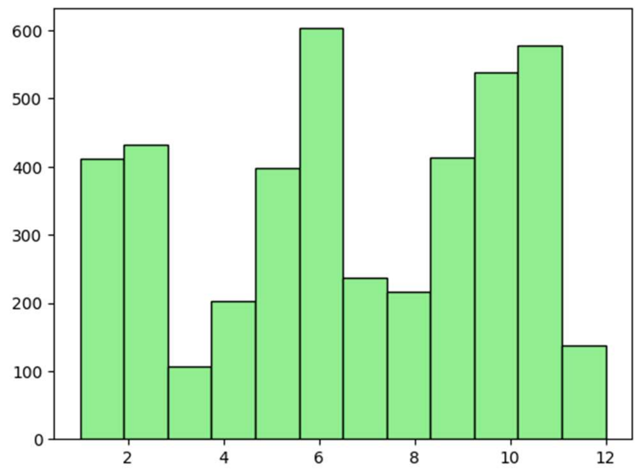


Figure 1 - Barplot of instances of plants

1	2	3	4	5	6	7	8	9	10	11	12
412	432	107	203	398	603	237	216	414	538	577	137

As we can see the dataset is severely unbalanced, with some plants having only 107 images and others over 600 images included in the dataset.

In the next step, the data has been visualized to get a better feel for the used images and be able to better determine which pre-processing steps should be applied to the images before using them in the neural networks. In the following, we can see a sample of 16 images taken from the test dataset depicting the leaves of different plant species.

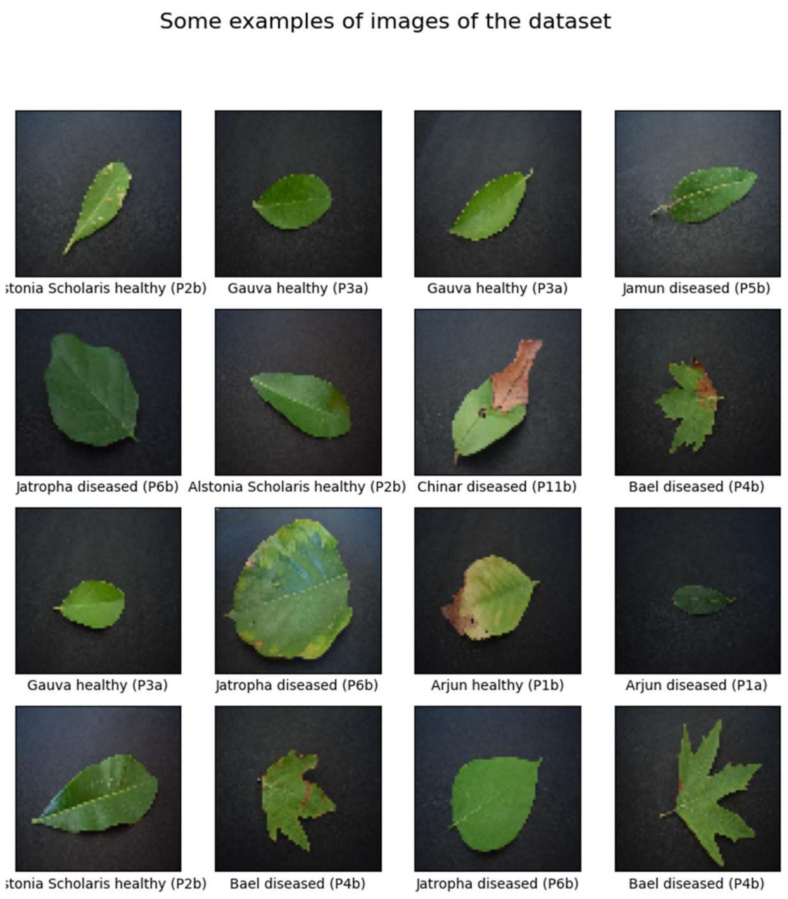


Figure 2 - Visualizing sample of leaves

3. Pre-processing

The pre-processing for this dataset was kept rather simple. Firstly, the images have been scaled down from their original dimension of 256x256 to 70x70 to save up on computational power and memory needed. While more fine-grained features and patterns could be found when using a higher resolution, the impact on the performance was too large to not scale them down. This scaling down reduced the load by a factor of over 13.

Next, the images had to be made usable for the neural network. This is done by transforming them into arrays by extracting the RGB values which lay between 0 and 255 for each pixel in each image and storing them in a 70x70x3 array. As a last step the images, or more precisely, their pixel values ranging from 0 to 255, were rescaled to values between 0 and 1 by dividing them by 255.

Data augmentation has been tested but did not improve the model results and was therefore discarded.

4. Convolutional Neural Network (CNN)

The convolutional neural network is the model of choice for the task at hand. It is a type of deep learning model specifically designed for image recognition and classification and consists of specific layers that enable it to learn and extract meaningful features from images. Without going into too much detail these layers are briefly explained in the following:

1. Input Layer: The input layer accepts the raw image data and passes it to the subsequent layers for processing.
2. Convolutional Layer: This layer performs the core operation of convolution, where it applies a set of learnable filters (also known as kernels) to the input image. Each filter scans the image, detecting different features by multiplying and summing the pixel values. This process creates feature maps that capture important patterns in the image.
3. Activation Layer: Following each convolutional layer, an activation function is applied elementwise to the feature maps. This introduces non-linearities into the network, allowing it to model complex relationships in the data. Due to its good performance the ReLU (Rectified Linear Unit) function has been used in this report.
4. Pooling Layer: Pooling layers help reduce the spatial dimensions of the feature maps, which reduces the computational complexity of the network. Max pooling has been used throughout all models. Here the maximum value in a small region is retained while discarding the rest. This helps capture the most important features and provides a good level of spatial invariance.
5. Fully Connected Layer: After several convolutional and pooling layers, fully connected layers are introduced. These layers connect every neuron in one layer to every neuron in the next layer. Fully connected layers are responsible for high-level reasoning in the network. They transform the features extracted by the previous layers into class probabilities to then classify the image.
6. Output Layer: The output layer provides the final predictions or outputs of the network. In the following models, the output layer consists of softmax activation that produces probabilities for the different classes.

5. Models

Different architectures have been tested and are shown including all results in the following.

5.1. Model

Model 0 was used as a baseline for the following model. Here a simple instance of a CNN was built. With only one convolution and pooling layer followed by two dense layers. Filter size has been chosen to be 5x5 in the convolution layer and 2x2 in the pooling layer respectively. A test accuracy of 73.64% has been achieved.

```
model_0 = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (5, 5), activation = 'relu', input_shape = (70, 70, 3)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(32, activation=tf.nn.relu),
    tf.keras.layers.Dense(22, activation=tf.nn.softmax)
```

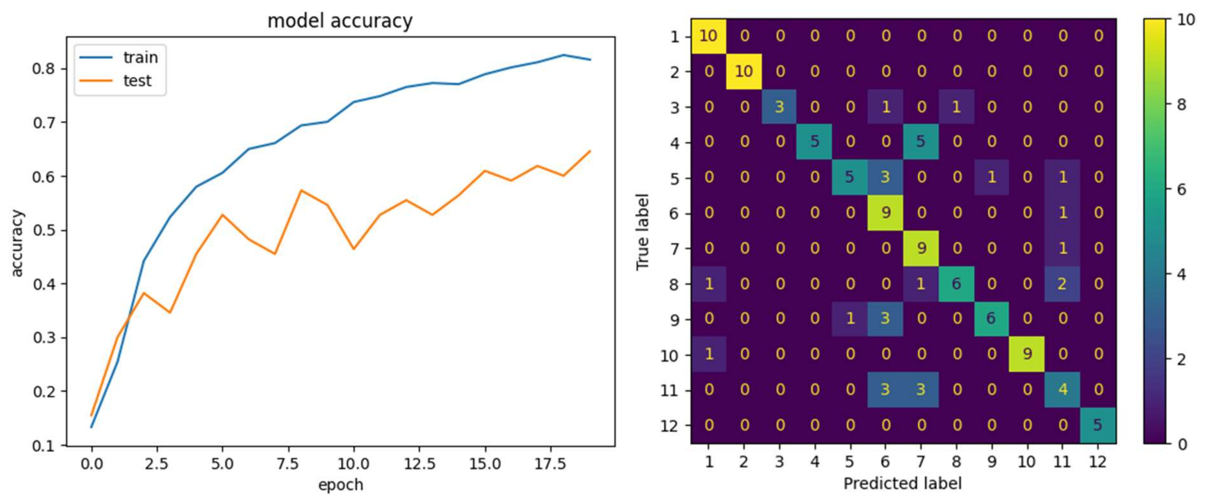


Figure 3 - Accuracy and Confusion matrix of Model 0

```
              precision    recall  f1-score   support
           1       0.83      1.00      0.91        10
           2       1.00      1.00      1.00        10
           3       1.00      0.60      0.75         5
           4       1.00      0.50      0.67        10
           5       0.83      0.50      0.62        10
           6       0.47      0.90      0.62        10
           7       0.50      0.90      0.64        10
           8       0.86      0.60      0.71        10
           9       0.86      0.60      0.71        10
          10       1.00      0.90      0.95        10
          11       0.44      0.40      0.42        10
          12       1.00      1.00      1.00         5
    accuracy                           0.74       110
   macro avg       0.82      0.74      0.75       110
weighted avg       0.80      0.74      0.74       110
```

5.2. Model 1

To test the impact of the filter size of the convolution layer, it has been reduced to 3x3 with respect to Model 0. This alone has improved the model tremendously and the 3x3 filter size will therefore be used in the next models as well. A surprisingly high overall test accuracy of 82.73% has been achieved with this simple model. As can be seen by the gap between train and test accuracy, overfitting is a problem which we will try to solve by introducing dropout layers in model 2.

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu', input_shape = (70, 70, 3)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(32, activation=tf.nn.relu),
    tf.keras.layers.Dense(22, activation=tf.nn.softmax)
])
```

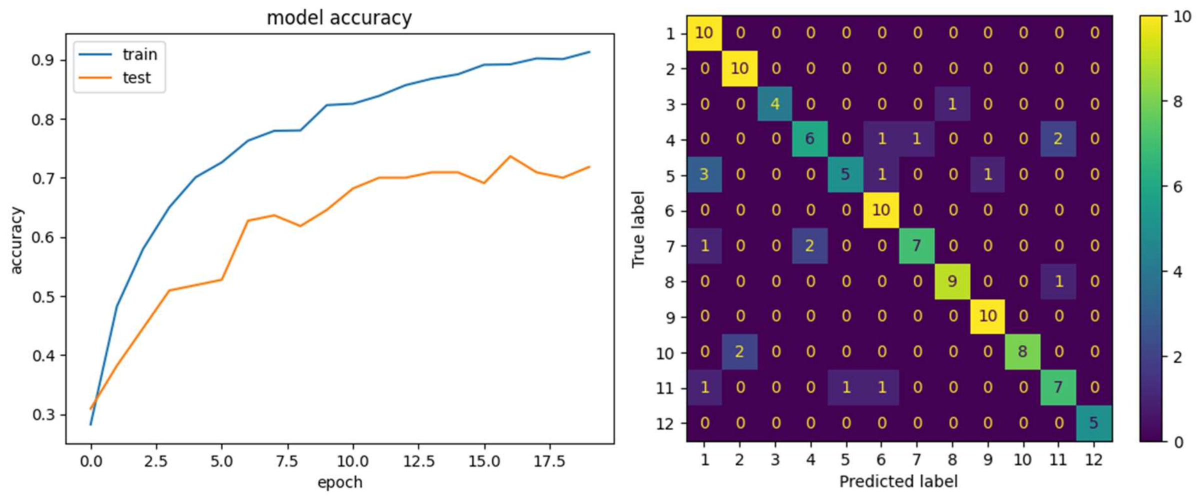


Figure 4 - Accuracy and Confusion matrix of Model 1

```
              precision    recall  f1-score   support
           1       0.67      1.00      0.80        10
           2       0.83      1.00      0.91        10
           3       1.00      0.80      0.89         5
           4       0.75      0.60      0.67        10
           5       0.83      0.50      0.62        10
           6       0.77      1.00      0.87        10
           7       0.88      0.70      0.78        10
           8       0.90      0.90      0.90        10
           9       0.91      1.00      0.95        10
          10       1.00      0.80      0.89        10
          11       0.70      0.70      0.70        10
          12       1.00      1.00      1.00         5
    accuracy                           0.83       110
   macro avg       0.85      0.83      0.83       110
weighted avg       0.84      0.83      0.82       110
```

5.3. Model 2

To increase the accuracy achieved with the first two models, three more convolution and pooling layers have been added to make a total of 4. The filter size has been lept at the value found best in model 1 3x3. As expected, the test accuracy increased by a lot to 90.91%. Also, dropout layers have been added after the pooling layers and one after the dense layer, to minimize overfitting and receive an overall better result on the test data.

```
model_2 = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu', input_shape = (70, 70, 3)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Conv2D(64, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Conv2D(128, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Conv2D(256, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation=tf.nn.relu),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(22, activation=tf.nn.softmax)
```

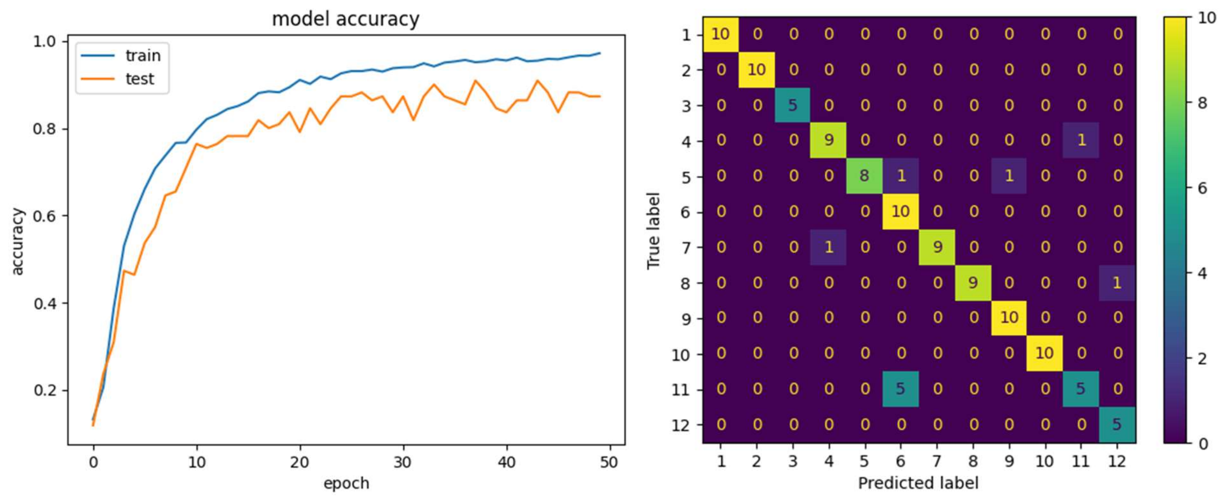


Figure 5 - Accuracy and Confusion matrix of Model 2

```
              precision    recall  f1-score   support
           1       1.00      1.00      1.00        10
           2       1.00      1.00      1.00        10
           3       1.00      1.00      1.00         5
           4       0.90      0.90      0.90        10
           5       1.00      0.80      0.89        10
           6       0.62      1.00      0.77        10
           7       1.00      0.90      0.95        10
           8       1.00      0.90      0.95        10
           9       0.91      1.00      0.95        10
          10       1.00      1.00      1.00        10
          11       0.83      0.50      0.62        10
          12       0.83      1.00      0.91         5
    accuracy                           0.91       110
   macro avg       0.93      0.92      0.91       110
weighted avg       0.93      0.91      0.91       110
```

5.4. Model 3

In this model, a batch normalization layer has been added after each pooling layer of the original structure of model 2. Batch normalization usually acts as an additional regularizer to dropout by adding some noise to the inputs, which can also help to reduce overfitting. Surprisingly, this resulted in a decrease in test accuracy of over 6% down to 84.55% for model 3.

```
model_3 = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu', input_shape = (70, 70, 3)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Conv2D(64, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Conv2D(128, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Conv2D(256, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation=tf.nn.relu),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(22, activation=tf.nn.softmax)
])
```

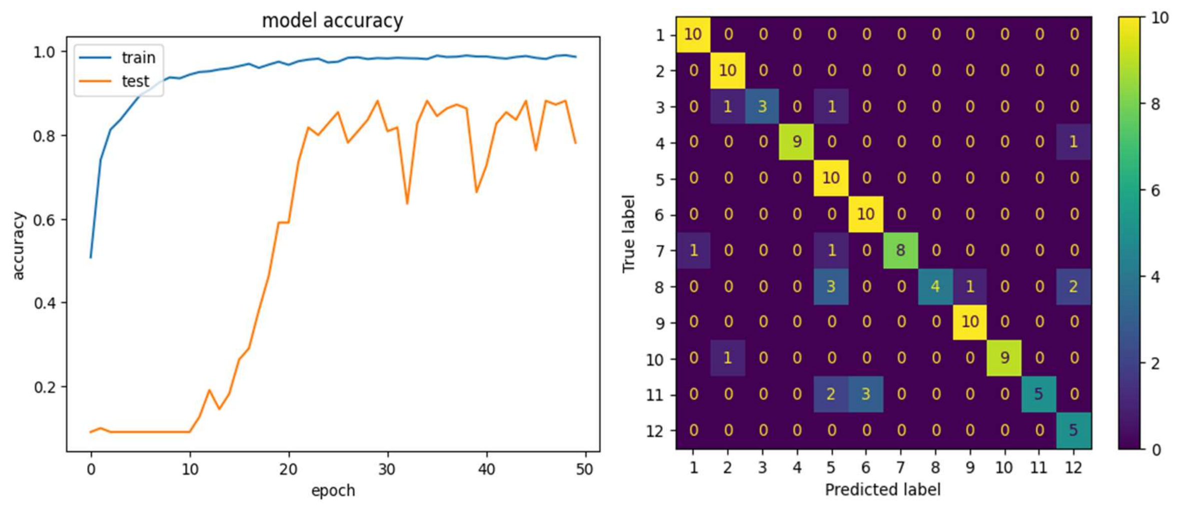


Figure 6 - Accuracy and Confusion matrix of Model 3

6. Conclusion

As could be seen in the report, even simple CNNs with few layers are able to provide reasonable results when the task at hand is image classification. After experimenting with the hyperparameters and architecture of the models a 90.91% accuracy was achieved.

Still, there are some issues in this analysis such as the small, provided sample size of 110 images for the validation and test set when compared to the training set which contained over 4000 images, which makes the results less robust due to the small support especially when monitored per class. Here a different split than the one provided could be chosen or data augmentation/generation techniques could have been applied to artificially increase sample size.