CentraleSupélec

Deep Learning Project Report

Cheikh Mohamed Vadhel - Aymen Zabori - Neil Achich

3A Mention SRI-Dominante SCOC

2022-2023

CentraleSupélec

Abstract

Deep learning is a powerful tool for making predictions from images. In this work, we demonstrate how a deep learning model can be used to predict whether a tomato is healthy or unhealthy based on images of its leaves. By training a convolutional neural network on a large dataset of tomato leaf images, we are able to achieve high accuracy in our predictions. Our model can be used by farmers and researchers to quickly and accurately assess the health of their tomatoes. This work has the potential to improve crop yield and reduce waste in the tomato industry.

Contents

1 Introduction

In this project, we aim to create a deep learning model that can predict the health of a tomato plant based on images of its leaves. By using a limited dataset and two different models - a Convolutional Neural Network (CNN) and a Transfer Learning model - we were able to train our model on a distant machine, achieving high accuracy in our predictions. In the following sections, we will describe the models we used, compare their performance, and provide our conclusions.

2 Framework

We chose to use TensorFlow Keras for our task because it is a powerful and user-friendly deep learning framework that allows us to easily build, train, and evaluate complex neural network models. TensorFlow is developed and maintained by Google, and it is widely used in research and industry for a variety of tasks including image classification, natural language processing, and time series analysis.

Keras is a high-level API that is built on top of TensorFlow, and it provides a simple and intuitive interface for defining, training, and evaluating deep learning models. It is designed to make it easy for researchers and developers to experiment with different model architectures and hyperparameters, allowing them to quickly and easily build and test new ideas.

TensorFlow Keras also offers many useful features and tools that are helpful for our task, such as built-in support for data augmentation, automatic differentiation, and the ability to easily save and load trained models.

3 Dataset

First, we started using the raw dataset that we obtained from kaggle but we faced a problem of "Class Imbalance". To prevent this problem, we have done some preprocessing regarding labels and their distribution, we obtained a collection of tomato leaf images which included a train set that was divided into two classes - 1000 healthy tomato leaf images and 1000 unhealthy tomato leaf images (vs 1000 healthy and 9000 unhealthy before). We then set aside a consequent number of additional tomato leaf images (10000) to use for testing our model later.
In order to prepare the dataset for training our model, we performed some preprocessing steps such as resizing the images and splitting the data into training and validation sets. We also applied data augmentation techniques to expand the size of our training dataset and improve the generalizability of our model. Once these steps were completed, we were ready to train our deep learning models.

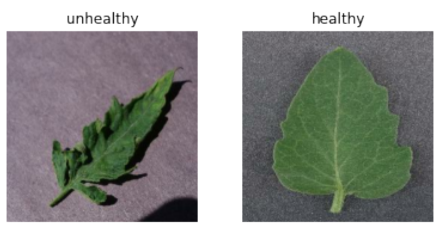

Figure 2: Images of two tomato leaves from the dataset

4 Convolutional Neural Networks Model

Convolutional Neural Networks (CNNs) are a type of deep learning model that is widely used for tasks involving image analysis and classification. This is because CNNs are able to automatically learn and extract high-level features from images, such as edges, corners, textures, and patterns, which are important for image recognition tasks.

CNNs are particularly well-suited for our task of predicting the health of a tomato plant based on its leaf images. This is because CNNs are designed to take advantage of the spatial structure of images, allowing them to learn features and patterns that are relevant to the task at hand. In the case of our tomato leaf images, a CNN can learn to identify important visual features such as the shape and color of the leaves, as well as any abnormalities or signs of disease.

Additionally, CNNs are known to perform well on a wide range of image classification tasks, making them a natural choice for our task. They are also highly scalable and can be trained on large datasets using a parallelized approach, which makes them well-suited for tasks that require high accuracy and fast performance.

Another advantage of using CNNs for our task is that they can be trained on a limited dataset. In our case, we have only 1000 healthy and 1000 unhealthy tomato leaf images to work with, which may not be sufficient for other machine learning algorithms to learn the underlying patterns in the data. However, CNNs are known to be able to learn effectively from small datasets, making them a good choice for our task.

Overall, the ability of CNNs to automatically learn high-level features from images, their scalability and performance on image classification tasks, and their ability to handle limited datasets make them the model of choice for our task of predicting the health of a tomato plant based on its leaf images.

4.1 Preprocessing layers

The first layers of our CNN model are responsible for preprocessing the input images. The first layer, **resize_and_rescale**, uses the **Resizing** and **Rescaling** layers from the TensorFlow Keras **image_preprocessing** module to resize and rescale the input images. The **Resizing** layer resizes the images to the specified size (in our case, 256), while the **Rescaling** layer scales the pixel values in the images from the range [0, 255] to the range [0, 1]. This is a common preprocessing step that is used to ensure that the input images are in a suitable format for the rest of the model.

The next layer, **data_augmentation**, uses the **RandomFlip** and **RandomRotation** layers from the **image_preprocessing** module to augment the input images. The **RandomFlip** layer randomly flips the images horizontally and vertically, while the **RandomRotation** layer randomly rotates the images by a small angle (in our case, 0.2 radians).

Data augmentation is a technique that is used to artificially expand the size of the training dataset by generating additional, slightly modified versions of the original images. This can help to improve the performance and generalizability of the model by reducing overfitting and making the model more robust to small variations in the data.

4.2 Core Layers

The model contains a sequence of convolutional, pooling, and dense layers that are connected in a feedforward manner.

The first layer in the sequence is a convolutional layer (**Conv2D**), which applies a set of convolutional filters to the input images. These filters are learned during training, and they are used to extract specific features from the images, such as edges, corners, and textures. The convolutional layer is followed by a max pooling layer (**MaxPooling2D**), which downsamples the output of the convolutional layer by taking the maximum value over a small spatial window. This reduces the size of the output and helps to make the model more efficient and less prone to overfitting.

The next two convolutional layers are similar to the first, but they use a larger number of filters and a larger stride to extract more complex features from the images. These layers are also followed by max pooling layers, which downsample the output of the convolutional layers.

After the convolutional layers, the model contains a flattening layer (**Flatten**), which converts the output of the convolutional layers into a 1D tensor. This is followed by a dropout layer (**Dropout**), which randomly drops some of the activations in the tensor in order to prevent overfitting.

The final layers of the model are two dense layers (**Dense**), which use the flattened output of the convolutional layers to make the final prediction. The first dense layer uses the ReLU activation function, which allows it to learn non-linear relationships in the data. The second dense layer uses the sigmoid activation function, which produces a probability score for each of the two classes (healthy and unhealthy). This allows the model to make a binary classification decision based on the predicted probabilities.

Overall, these layers form the core of our CNN model, and they are responsible for learning the features and patterns in the input images that are relevant to the task of predicting the health of a tomato plant.

4.3 Mathematical description

Let X be the input image tensor with dimensions $n \times h \times w \times c$, where n is the number of images, h and w are the height and width of the images, and c is the number of channels (e.g. 3 for RGB images).
$X' = resize\_and\_rescale(X)$
$X'' = data\_augmentation(X')$
$Z_1 = conv2d(X'', 32, (3,3), activation = relu)$
$Z_2 = maxpool(Z_1, (2,2))$
$Z_3 = conv2d(Z_2, 64, (3,3), activation = relu)$
$Z_4 = maxpool(Z_3, (2,2))$
$Z_5 = conv2d(Z_4, 64, (3,3), activation = relu)$
$Z_6 = flatten(Z_5)$
$Z_7 = dropout(Z_6, rate = 0.5)$
$Z_8 = dense(Z_7, 64, activation = relu)$
$Z_9 = dense(Z_8, 2, activation = sigmoid)$

This output tensor Z_9 contains the predicted probabilities of the input images belonging to the healthy and unhealthy classes.

The sigmoid function, also known as the logistic function, is a mathematical function commonly used in machine learning models. It is defined as:

$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

The sigmoid function maps any real-valued input to a value between 0 and 1. This makes it useful for modeling probabilities, as the output of the function can be interpreted as the probability of an event occurring.

The rectified linear unit (ReLU) is a commonly used activation function in neural network models. It is defined as:

$$relu(x) = max(0, x)$$

where x is the input to the activation function.

The ReLU function outputs the input x if $x > 0$ and outputs 0 if $x \leq 0$. This means that the ReLU function only passes through positive values, effectively thresholding negative values at 0.

4.4 Compilation

When compiling a deep learning model, we need to specify an optimizer and a loss function. The optimizer is responsible for updating the model weights during training, while the loss function measures the difference between the model's predictions and the ground truth labels.

For our model, we chose the Adam optimizer, which is a popular and effective optimization algorithm for training deep learning models. Adam combines the advantages of the AdaGrad and RMSProp algorithms, and it has been shown to converge quickly and reliably on a wide range of tasks. Adam is also well-suited for training large and complex models, such as our CNN, because it can efficiently update the model weights using mini-batch gradient descent.

Adam (Adaptive Moment Estimation) is a gradient-based optimization algorithm used to train machine learning models. It is an extension of the stochastic gradient descent (SGD) algorithm that uses exponentially weighted averages of the gradients and squared gradients to adaptively adjust the learning rates of individual parameters.

Mathematically, Adam can be written as follows:
Let g_t be the gradient of the loss function with respect to the model parameters at timestep t, and let m_t and v_t be the first and second moments of the gradient, respectively. The Adam algorithm updates the model parameters at timestep t as follows:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$
$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$
$$\theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{\hat{v}_t}+\epsilon} \hat{m}_t$$

where α is the learning rate, β_1 and β_2 are the decay rates for the first and second moments, respectively, and ϵ is a small constant to prevent division by zero. The Adam algorithm uses these equations to adaptively adjust the learning rates of individual parameters based on the first and second moments of the gradients.

For our loss function, we chose the Sparse Categorical Crossentropy loss. This loss function is commonly used for tasks involving multi-class classification, such as our task of predicting the health of a tomato plant.

The Sparse Categorical Crossentropy loss measures the difference between the model's predicted probabilities and the ground truth labels, and it is designed to be used with integer labels rather than one-hot encoded labels. This makes it a good choice for our task, where the labels are 0 (healthy) and 1 (unhealthy). It can be written mathematically as follows:
Let Z be the output tensor of the CNN model with dimensions $n \times k$, where n is the number of images and k is the number of classes. Let Y be the ground truth tensor with dimensions $n \times 1$, containing the true class labels of the input images.

The SparseCategoricalCrossentropy loss function measures the difference between the predicted probabilities Z and the ground truth labels Y. It computes the cross-entropy loss for each sample in the batch, then averages the losses over the batch. This can be represented mathematically as:

$$L = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{k} -y_{i,j} \log z_{i,j}$$

where $z_{i,j}$ is the predicted probability of image i belonging to class j, and $y_{i,j}$ is the ground truth label of image i (1 if image i belongs to class j and 0 otherwise).

4.5 Training

To train our convolutional neural network (CNN) model for 20 epochs with batch training and a validation data set equaling 0.1 of the original dataset, we first split the dataset of 2000 images into small batches of a specified size. We then set aside a portion of the training data as a validation set, which will be used to evaluate the model's performance during training.

Next, we used the fit method of the Keras model to train the model for 20 epochs. This ran the training process for 20 iterations, updating the model's weights and biases after each batch of data to minimize the loss. The fit method also evaluated the model's performance on the validation set after each epoch and recorded the accuracy and loss.

Throughout the training process, we monitored the model's performance using the history attribute of the Keras model, which contains a record of the training and validation accuracy and loss after each epoch. This allowed us to track the model's learning and ensure that it was training effectively.

4.6 Results

The final loss rate of 0.27 and accuracy of 0.95 indicate that the model was able to learn effectively from the training data and make accurate predictions on the training set. The validation loss of 0.18 and accuracy of 0.97 suggest that the model was able to generalize well to unseen data and make accurate predictions on the validation set.

However, when testing on other test data for 200 equally distributed healthy and unhealthy tomato leaves images. The model got 62 correct unhealthy images and only 50 correct healthy images. This shows that the model performance is poor and uncertain on new data. A solution for that problem is to train the model on a much larger dataset.
We can see that even with a high accuracy model, when the loss rate is consequent the model will most probably perform poory with uncertainty on new data.

The use of the Adam optimizer and the binary cross-entropy loss function likely

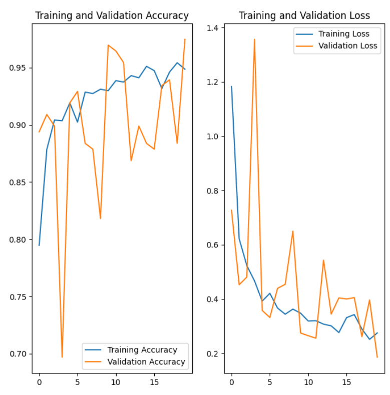

Figure 3: Accuracy and Loss

contributed to the model's ability to learn effectively and achieve good performance on the training, validation sets. The Adam optimizer is a popular choice for training neural networks and has been shown to achieve good performance on a wide range of tasks. The binary cross-entropy loss function is well-suited for classification tasks with two classes, such as healthy and unhealthy tomato leaves, and can help the model learn to make accurate predictions.

There are several ways in which the performance of this model could be improved in terms of the metrics of loss and accuracy. Some potential approaches include:

Increasing the number of epochs: Training the model for a larger number of epochs could allow it to learn more from the data and potentially improve its performance on the training, validation, and test sets.

Using a larger batch size: Increasing the batch size could allow the model to see more data at each training step, potentially improving its performance.

Tuning the model's hyperparameters: Adjusting the values of the model's hyperparameters, such as the learning rate, regularization strength, and number of filters in the convolutional layers, could improve the model's performance and enable it to learn more effectively from the data.

Adding more data to the training set: Increasing the size of the training set could provide the model with more examples to learn from and potentially improve its performance.

5 Transfer Learning

Transfer learning with a pre-trained ResNet50 model can be used to create a model that is better than the previously used model in several ways.

First, the ResNet50 model is a highly-efficient convolutional neural network (CNN) that has been trained on a large dataset and has achieved state-of-the-art performance on a variety of tasks. By starting with a pre-trained ResNet50 model, we can leverage the knowledge and features learned by the model and use them as a starting point for our own model. This can save time and resources compared to training a model from scratch, and can often result in improved performance.

Second, the ResNet50 model has been trained on a large dataset and has learned to extract generic features from images that are useful for a wide range of tasks. By using transfer learning, we can fine-tune the pre-trained ResNet50 model on the task of identifying healthy and unhealthy tomato leaves, allowing it to learn task-specific features and improve its performance on this task.

Third, the ResNet50 model is highly efficient and can be trained quickly, even on a limited computation budget. This makes it an ideal candidate for transfer learning, as it allows us to train a model with good performance quickly and without the need for extensive computation resources.

5.1 Layers

The model consists of two parts: the pre-trained ResNet50 model, which is used as a base, and a set of additional layers that are added on top of the base model.

The pre-trained ResNet50 model is imported from the TensorFlow Keras applications library and initialized with the input shape of the images and the 'imagenet' weights. The model's layers are then frozen, so that they will not be updated during training. This allows us to use the pre-trained ResNet50 model as a feature extractor, leveraging the knowledge and features learned by the model on the ImageNet dataset.

Next, a set of additional layers is added on top of the base model. These layers include a resize and rescale layer, a data augmentation layer, and several convolutional, pooling, and dense layers. These layers are trained from scratch on the task of identifying healthy and unhealthy tomato leaves, allowing the model to learn task-specific features and improve its performance.

5.2 Compilation

The model uses the same optimizer and loss metric as the previous one.

5.3 Training

The model was trained on the same dataset as the previous one but with 5 epochs. In general, more epochs can be beneficial for training a model, as they allow the model to see more data and potentially learn more from the training set. However, training for too many epochs can result in overfitting, where the model learns the noise and details of the training set too well and does not generalize well to unseen data.

Therefore, it is important to carefully select the number of epochs for training a model, balancing the need to learn from the training data with the risk of overfitting.

5.4 Results

The training and validation accuracy of this model are quite high, which suggests that it is performing well on the training and validation data. The relatively low validation loss (0.0879) and high validation accuracy (0.9948) also indicate that the model is not overfitting to the training data, as overfitting would typically result in a low validation accuracy and high validation loss. The test accuracy rate is around 0.99.

One potential reason for the model's high accuracy is the use of transfer learning. By using a pre-trained model, the model is able to benefit from the knowledge and features learned by the pre-trained model on a large dataset, such as ImageNet. This can help the model learn more effectively and improve its performance on the target task.

The model was tested on 100 images of unhealthy tomato leaves and a 100 images of healthy tomato leaves. It got 99 correct unhealthy images and 87 correct healthy images. This gives far better accuracy than the previous CNN model and shows how performant transfer learning can be for images label prediction.

Additionally, the use of data augmentation and dropout regularization may also help prevent overfitting and improve the model's performance.

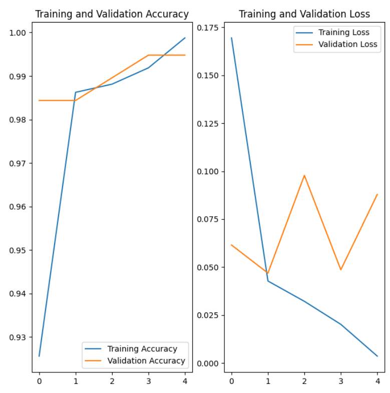

Figure 4: Accuracy and Loss

6 Conclusion

Training a model on a small dataset can pose some challenges, as the model may not have enough data to learn effectively and may overfit to the training data. In this case, both the CNN model and the transfer learning model are trained on a relatively small dataset of only 2000 images, which means they may not perform as well as they would if trained on a larger dataset.

However, it's possible that the use of data augmentation and regularization techniques may help mitigate some of the potential issues associated with training on a small dataset. Data augmentation artificially increases the size of the training dataset by applying random transformations to the training images, which can help the model generalize better. Regularization techniques, such as dropout, can also help prevent overfitting by encouraging the model to learn more robust features.

Moreover, if the model performs well on the test data, it may be an indication that it is able to generalize well to new examples, despite being trained on a small dataset. In this case, it's possible that the model may still be useful for the intended task, even if it was trained on a small dataset.

In general, the model with the higher accuracy and lower loss on the test data is likely to be the better model, as these metrics are commonly used to evaluate a model's performance.

However, there are other factors that can affect a model's performance and suitability for a given task. For example, the complexity of the model, the amount of training data, and the use of regularization techniques can all impact a model's performance.

In our case, the transfer learning model has an extremely higher accuracy and a much lower loss on the test data compared to the CNN model, it may be considered the better model. Moreover, the transfer learning model uses a pre-trained model (ResNet50), it is more potentially able to generalize to new data and prevent overfitting.

A step forward to the project, is to train the models on a much larger dataset while choosing manually the distribution of the data to avoid class imbalance problems and to study also the performances of other methods such as RNNs, GANs and Ensemble Learning that can be interestingly applied to our task.

The code for the project can be found in https://github.com/cheikhmc/deep_learning_project where the model will be implemented in a Flask app, The model was trained on a shell script in a distant machine-because of computing resources constraints-and saved on disk.

7 Bibliography

Deep Learning Lectures, Prof. Pablo Piantanida (CNRS, CentraleSupelec), Director of the International Laboratory on Learning Systems (ILLS), McGill - ETS - MILA - CNRS - Université Paris-Saclay - CentraleSupelec(Canada)

https://www.kaggle.com/datasets/kaustubhb999/tomatoleaf

https://www.tensorflow.org/api_docs/python/tf/keras

https://www.informatica.si/index.php/informatica/article/viewFile/2828/1433

https://arxiv.org/pdf/1511.08458.pdf

https://arxiv.org/pdf/1412.6980.pdf