Final Project Report

Motivation and Objective (Problem, Challenge):

Our model will detect plant disease in early stages by their images. Goal is that we need to build a model, which can classify between healthy and diseased crop leaves and also if the crop has any disease, predict which disease is it

Related work and originality:

Now we have dataset from kaggle (<https://www.kaggle.com/datasets/emmarex/plantdisease>). Here different type of the vegetables, fruits, flowers, with healthy ones and with some afflictions.

Design architecture:

This model is a convolutional neural network (CNN) architecture designed for image classification. It consists of several layers that perform convolution, activation, pooling, batch normalization, dropout, and dense operations. Here is a breakdown of the layers: Convolutional neural network with 6 major layers.

Input layer: The input shape of the model is 256x256 with 3 channels.

Convolutional layer: The model starts with a convolutional layer with 32 filters of size 3x3 and padding "same". This layer applies 32 different filters to the input image to extract 32 different features.

Activation layer: An activation function (ReLU) is applied to the output of the convolutional layer to introduce non-linearity into the model.

Batch normalization layer: A batch normalization layer is added after the activation layer to normalize the output and improve the stability of the model during training.

Max pooling layer: A max pooling layer is added to reduce the spatial dimensions of the output of the previous layer by taking the maximum value within each 3x3 patch.

Dropout layer: A dropout layer is added to randomly drop out 25% of the activations in the previous layer to prevent overfitting.

Two more sets of convolutional, activation, batch normalization, max pooling, and dropout layers are added with increasing numbers of filters (64 and 128).

Flatten layer: The output of the last max pooling layer is flattened into a one-dimensional vector to be fed into a dense layer.

Dense layer: A dense layer with 1024 neurons is added to perform a linear transformation on the output of the previous layer.

Activation layer: An activation function (ReLU) is applied to the output of the dense layer to introduce non-linearity.

Batch normalization layer: A batch normalization layer is added to normalize the output of the previous layer.

Dropout layer: A dropout layer is added to randomly drop out 50% of the activations in the previous layer to prevent overfitting.

Output layer: A dense layer with n\_classes neurons (where n\_classes is the number of classes in the classification task) is added to produce the final output probabilities. An activation function (softmax) is applied to ensure that the output probabilities sum up to 1.0.

Detailed algorithm or functions:

We decided to make with multiple convolutional layers. Layers consist Conv2d, Batch Normalization, Max Pooling and Dropout (for lowering the overfitting). We used Max Pooling 3 times since the images are too small.

The function that loads images and their corresponding labels from a hierarchical directory structure containing plant folders and their respective disease subfolders. It iterates through the directory structure, filtering out any ".DS\_Store" files, and collects the first 200 image files (with ".jpg" or ".JPG" extension) from each disease subfolder. The images are then converted into arrays using the convert\_image\_to\_array() function (not provided in the code snippet) and appended to the image\_list, while the corresponding disease labels are appended to the label\_list.

Coding: https://github.com/MereyPolatkhan/plant\_disease\_project\_ML\_course/blob/master/code.ipynb

model = Sequential()

chanDim = -1

model.add(Conv2D(32, (3, 3), *padding*="same",*input\_shape*=(256, 256, 3)))

model.add(Activation("relu"))

model.add(BatchNormalization(*axis*=chanDim))

model.add(MaxPooling2D(*pool\_size*=(3, 3)))

model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), *padding*="same"))

model.add(Activation("relu"))

model.add(BatchNormalization(*axis*=chanDim))

model.add(Conv2D(64, (3, 3), *padding*="same"))

model.add(Activation("relu"))

model.add(BatchNormalization(*axis*=chanDim))

model.add(MaxPooling2D(*pool\_size*=(2, 2)))

model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3), *padding*="same"))

model.add(Activation("relu"))

model.add(BatchNormalization(*axis*=chanDim))

model.add(Conv2D(128, (3, 3), *padding*="same"))

model.add(Activation("relu"))

model.add(BatchNormalization(*axis*=chanDim))

model.add(MaxPooling2D(*pool\_size*=(2, 2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(1024))

model.add(Activation("relu"))

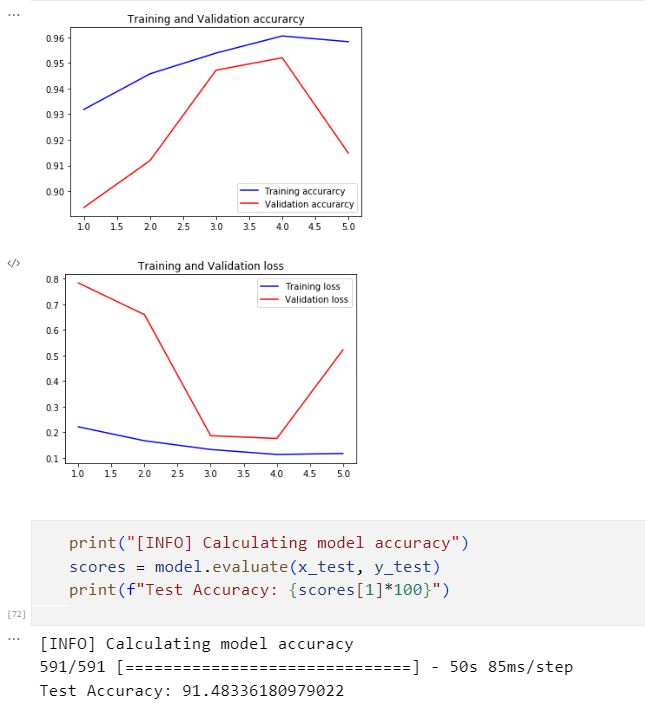
model.add(BatchNormalization())

model.add(Dropout(0.5))

model.add(Dense(n\_classes))

model.add(Activation("softmax"))

Results and performance evaluation:



Conclusion:

To sum up, it is better to put convolutional layers with lower number of filters before convolutional layers with higher numbers. Max Pooling helps to highlight important image details while reducing training time. Batch Normalization normalizes values between hidden layers and thus improves generalization. Due to the fact that pool reduces the image, we decided to alternate layers with pool and without pool. ReLu is very good as it is an easy to compute function. We used softmax, since sigmoid gave worse results and decided to output not one neuron, but the number of classes and softmax selects the appropriate one based on training.

In terms of the architecture, this CNN model has a deep architecture with multiple convolutional layers to extract features from the input images. The use of activation functions, batch normalization, and dropout layers are also included to introduce non-linearity, improve the stability of the model, and prevent overfitting. Additionally, the inclusion of a dense layer with 1024 neurons helps the model to learn higher-level representations of the input data.

Overall, this architecture is a well-designed CNN model that demonstrates the effectiveness of deep learning for image classification tasks.

References:

https://www.geeksforgeeks.org/introduction-convolution-neural-network/

<https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/>

<https://kaggle.com/datasets/>