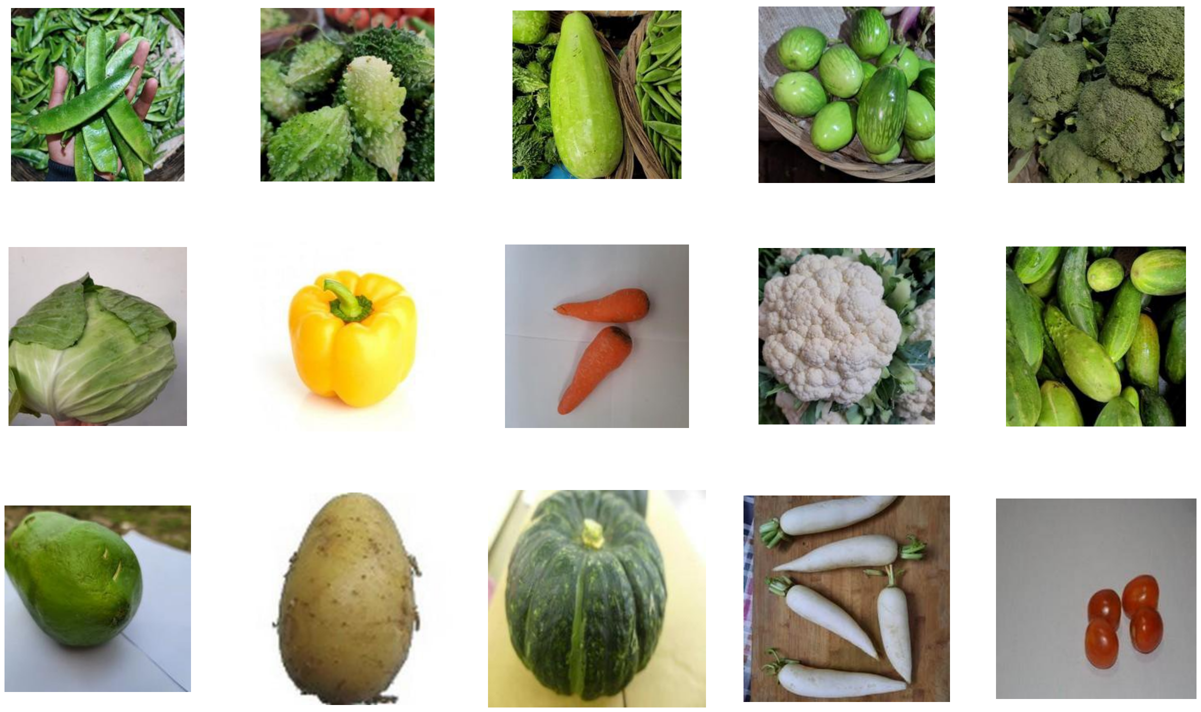
Convolutional Neural Networks 2D wiht tf.keras

The project aims to achieve a classification model using TensorFlow Keras. The model has been trained with the database containing 15 classes ['Bean', 'Bitter_Gourd', 'Bottle_Gourd', 'Brinjal', 'Broccoli', 'Cabbage', 'Capsicum', 'Carrot', 'Cauliflower', 'Cucumber', 'Papaya', 'Potato', 'Pumpkin', 'Radish', 'Tomato'], 1500 training dataset, 3000 validatin dataset and 3000 test dataset, taken from: 'https://www.kaggle.com/api/v1/datasets/download/misrakahmed/vegetable-image-dataset'

The project contains:

- Describes the methods and functions used
- Functionality diagrams
- Diagrams of results obtained
- Project code



Data collection method

A straightforward way to create the dataset is to use the image_dataset_from_directory method, which is found in the Keras TensorFlow library tf.keras.preprocessing.image_dataset_from_directory. This method creates a tensor of the form [[[[image][label]]] [[image][label]] [[[image][label]]] .....], and the required structure has the form

Dataset_directory/

Label_a/	Label_b/	Label_c/	… /
a_image_1.jpg	b_image_1.jpg	c_image_1.jpg	⋯
a_image_2.jpg	b_image_2.jpg	c_image_2.jpg	⋯
a_image_3.jpg	b_image_3.jpg	c_image_3.jpg	⋯
⋮	⋮	⋮	⋮

Data Normalization

An important step in data pre-processing is feature normalization, which is a method of scaling the features before entering them into a model. Properly, normalization means scaling the data to be analyzed into a specific range [0,0 ... 1]. The purpose of this method is to facilitate the model operation and to find a common denominator in terms of the amplitude of the different datasets that are composited. The normalization equation is:

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Separating data into features and labels

This method is used if the dataset has the form described above and the input to the model requires separating the features from the label, preserving the order.

Sequential()

The Sequential class in Keras allows rapid prototyping of machine learning models by sequentially superimposing layers. Sequential models work by layer stacking, i.e., each layer takes the output of the previous layer as input. To create a sequential model in Keras, you can create a list of constructor statements or add layers incrementally using the add() method.

Conv2D()

tf.keras.layers.Conv2D from the TensorFlow Keras libraries is a convolutional element with which conventional neural networks are built. The explanation of these network models is explained in a previous project https://github.com/tot-alin/CNN-with-tf.nn

Conv2D requires some minimal settings Conv2D(filters=nr.maps out, kernel_size=(height,width), strides=moving on (height,width), padding= same or valid, activation=chosen activation function)

Pool

Pooling is a method of extracting important features from a feature matrix (feature map) by reducing its size. The description of the functionality can be found at https://github.com/tot-alin/CNN-with-tf.nn/blob/main/CNN_tf_nn.pdf

The settings for MaxPooling2D(pool_size=(height, width), strides=moving on (height, width), padding=same or valid), used in this project, extract the maximum values from the moving window.

Flatten()

This class transforms an m×n-dimensional matrix or an i×j×k×⋯ tensor into a vector (one-dimensional matrix) 1×(m*n) or 1×(i*j*k*⋯). The role of this transform in this project is to bridge the CNN layer and the Dense layer.

Dense()

tf.keras.layers.layers.Dense() is a fully connected layer where each neuron in the layer is connected to the next layer, more fully described in https://github.com/tot-alin/Multi-Layer-Perceptron-with-NumPy.

Minimum Dense class setting (units = number of neurons in layer, activation = chosen activation function).

Activation function

The activation functions in a neural network have the role of changing the linear character of the output produced by layers of perceptrons, CNN, etc. In this project, there are two types of networks: ReLU and SoftMax.

SoftMax is the activation function that converts a raw feature vector from the neural network into a vector expressing the probability corresponding to the input. The computational relation$SoftMax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{J} e^{z_j}}$ $for\ i = 1...J$, z_i - the output of the previous layer in the network, J - number of classes, e^{z_i} - represents the exponential of the z_i, $\sum_{j=1}^{J} e^{z_j}$ - the sum of exponentials across all classes.

The function ReLU (rectified linear unit) is a linear function on the range of positive values and on the range of negative values or 0; this function outputs the value 0. Function expression $ReLU(z) = max(0,z)$ where z is the input.

Dropout()

Dropout() is a technique for decreasing the overfitting of neural networks by randomly removing a random amount of information from the matrix that passes through this method. This functionality is present during the training period; in the case of evaluation or prediction, it stops automatically. Setting tf.keras.layers.Dropout(elimination coefficient between 0 and 1)

Loss

The loss function, also called the error function, quantifies the difference between the predicted outcome of a machine learning model and the actual target values. This function underlies the machine learning process.

The error function used in this project is categorical cross-entropy, used in multi-class classification problems, i.e., more than two. It measures the difference between the probability distribution vector (SoftMax) predicted by the model and the true label with values of 0 and 1. Function equation $Loss_{CCE}(y,y') = -\sum_{i=1}^{C} y_i * log\ (y_i')$ where $Loss_{CCE}(y,y')$ - is the categorical cross-entropy loss, y_i - is the true label (0 or 1 for each class) from the one-hot encoded target vector, y_i' – is the predicted probability for classi.

Metric

This is a method for evaluating machine learning models that indicates a quantitative value of the efficiency of the models. Regardless of the type of problem, classification, continuous value prediction, or clustering, the correct selection of the evaluation method tells us how well the model accomplishes its goals.

Accuracy is a fundamental method used to evaluate the performance of a classification model and expresses the proportion of correct items to the total number of items. The equation can be expressed as $accuracy = \frac{1}{total\ n}(argmax(y_n) = argmax(y_n'))$ where $argmax(y_n)$ – indicates the maximum position in the target result vector and $argmax(y_n')$ – indicates the maximum position in the outcome vector predicted by the model.

Optimizer

Optimizers are algorithms used to minimize the loss function or to streamline the process. These mathematical functions relate to model learning parameters, i.e., weights, gradients, and errors. Specifically, these optimizers help to modify the weights and learning rate.

Compiling a Model

The compile() function creates and configures the model for the training and evaluation process. By calling this function, the model encapsulates the optimizer, loss, and metrics. If we omit the compile() function and call one of the fit() or evaluate() functions, an error will occur.

Model fit()

The fit() function is used for training a machine learning model using a fixed number of epochs. During training, the function adjusts trainable model parameters in order to minimize the loss function.

The objects returned by this class are the loss and metric values recorded during the training. model.fit(x_train, y_train, batch_size, epochs, validation_data=(x_val, y_val))

Model evaluate()

The evaluate() function is used to evaluate a trained model. It returns the loss and metric value. Generally used with dataset_test. evaluate(test_x, test_y)

Confusion Matrix For Multi-class Classification

The confusion matrix is a table used to express how good a classification model is. It compares the predictions made with the actual results and shows where it was right or wrong. This helps you understand where the model is wrong so you can improve it.

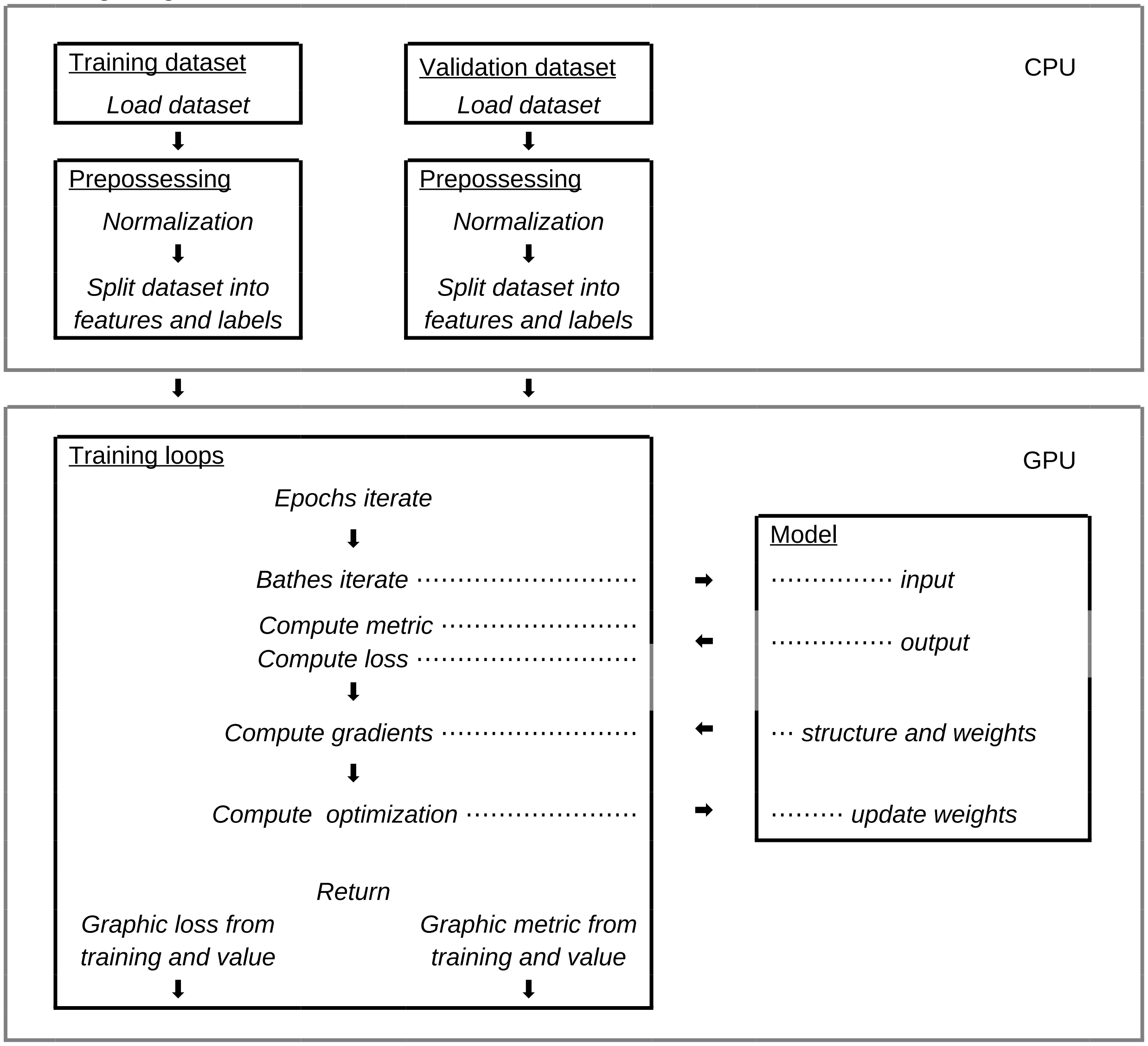
The multi-class confusion matrix concept is similar to the binary class matrix. The columns represent the original or expected class distribution, and the rows represent the classifier's predicted or output distribution.

Model description

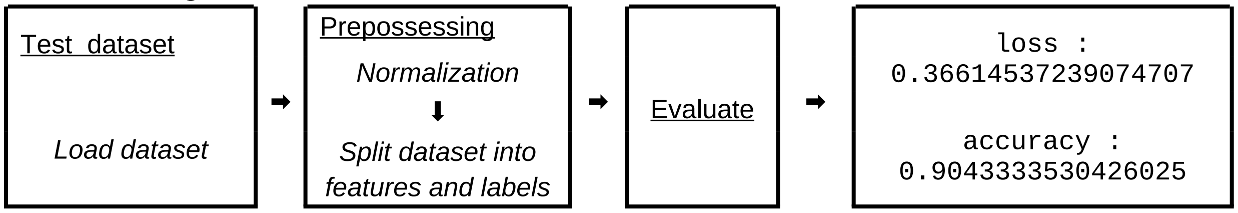
This model is a model that realizes the classification of objects and is composed of CNN layers and a fully connected layer. The dataset is taken from https://www.kaggle.com/api/v1/datasets/download/misrakahmed/vegetable-image-dataset and contains the train dataset - 15000 files belonging to 15 classes, the validation dataset - 3000 files belonging to 15 classes, and the test dataset 3000 files belonging to 15 classes.

To optimize computational resources, the project processing was performed in two compute units as follows. The pre-processing of the datasets was realized on the CPU unit, and the dataset remained stored in RAM (approx. 25 Gbi), and the model was trained on the GPU unit.

Training diagram



Evaluate diagram



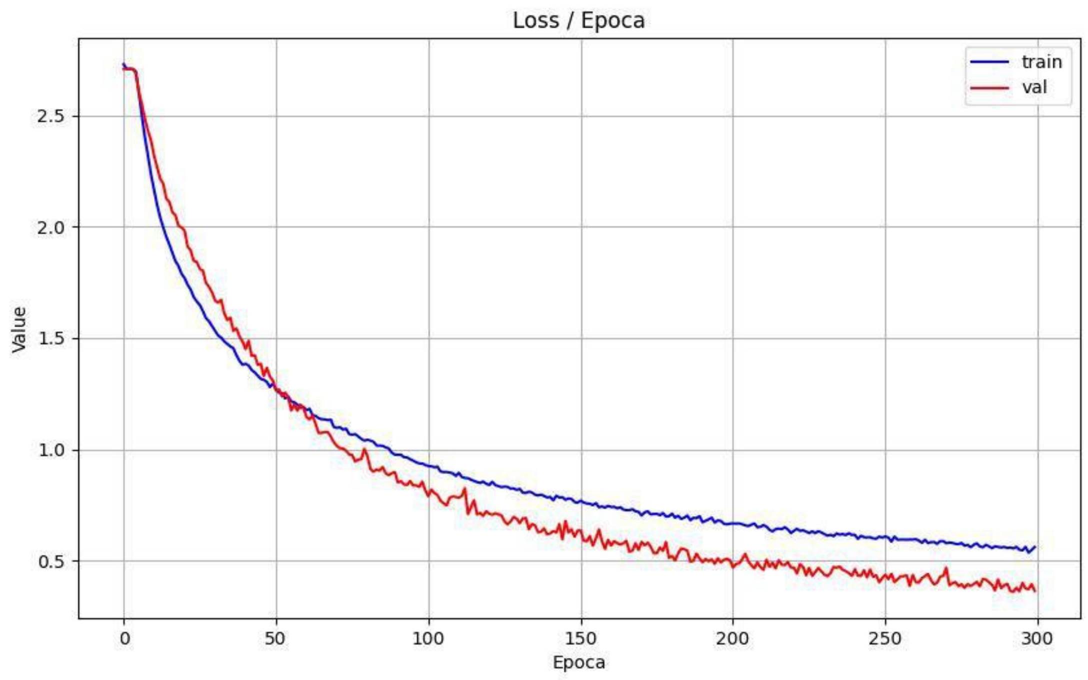
```
loss :
0.36614537239074707

accuracy :
0.9043333530426025
```

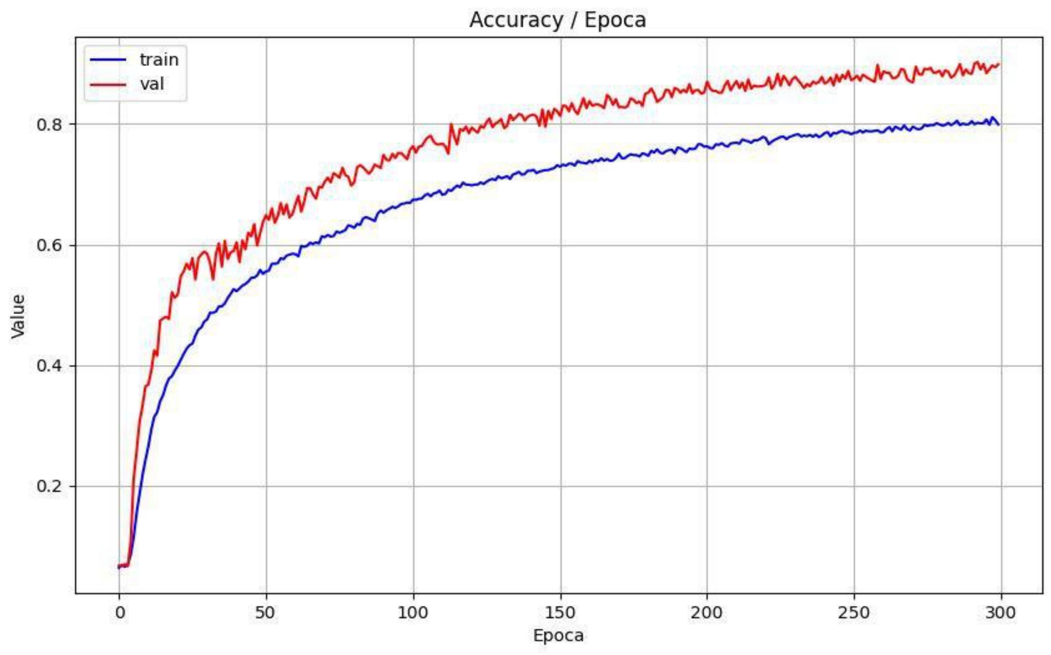
Model diagram

Input	224X224X3	
Conv2D	224X224X16	kernel_size = (3, 3), strides = (1, 1)
ReLu	224X224X16	
MaxPooling2D	112X112X16	pool_size = (2, 2),strides = (2, 2)
Dropout	112X112X16	rate = 0.5
Conv2D	112X112X16	kernel_size = (3, 3), strides = (1, 1)
ReLu	112X112X16	
MaxPooling2D	65X65X16	pool_size = (2, 2),strides = (2, 2)
Dropout	65X65X16	rate = 0.5
Conv2D	65X65X16	kernel_size = (3, 3), strides = (1, 1)
ReLu	65X65X16	
MaxPooling2D	28X28X16	pool_size = (2, 2),strides = (2, 2)
Dropout	28X28X16	rate = 0.5
Conv2D	28X28X16	kernel_size = (3, 3), strides = (1, 1)
ReLu	28X28X16	
MaxPooling2D	14X14X16	pool_size = (2, 2),strides = (2, 2)
Dropout	14X14X16	rate = 0.5
Flatten	3136	14X14X16=3136
Dense	32	
ReLu	32	
Dropout	32	rate = 0.5
Dense	15	
SoftMax	15	

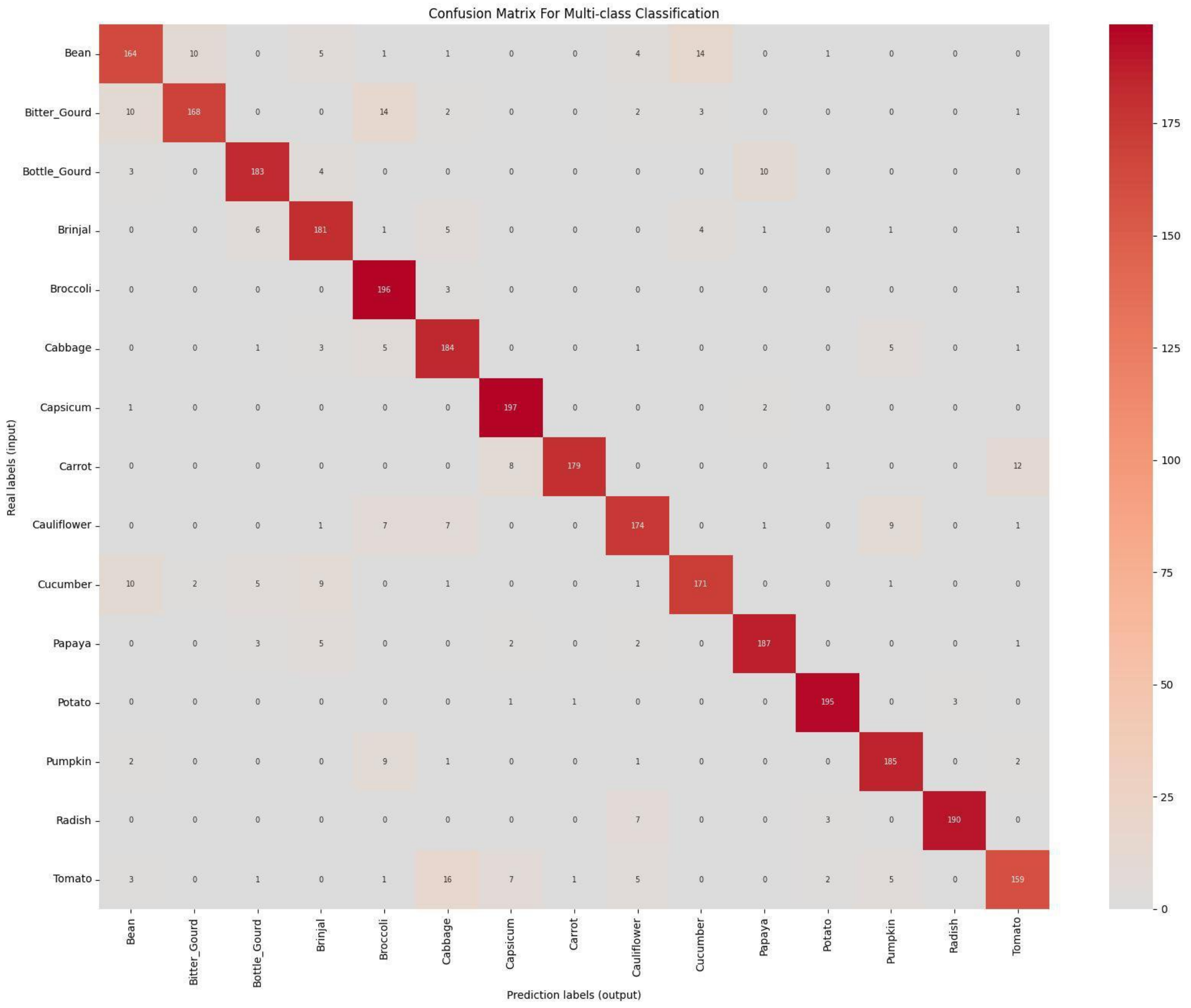
Loss diagram for training and validation datasets



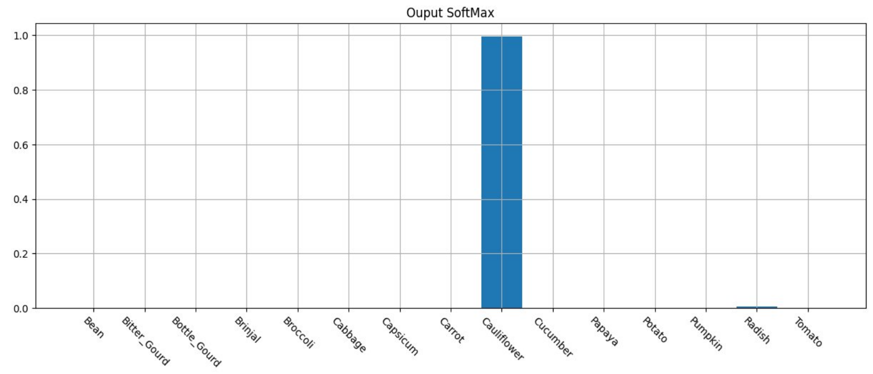
Accuracy diagram for training and validation datasets



Confusion Matrix for Multi-class Classification diagram



	Bean	Bitter_Gourd	Bottle_Gourd	Brinjal	Broccoli	Cabbage	Capsicum	Carrot	Cauliflower	Cucumber	Papaya	Potato	Pumpkin	Radish	Tomato
Bean	164	10	0	5	1	1	0	0	4	14	0	1	0	0	0
Bitter_Gourd	10	168	0	0	14	2	0	0	2	3	0	0	0	0	1
Bottle_Gourd	3	0	183	4	0	0	0	0	0	0	10	0	0	0	0
Brinjal	0	0	6	181	1	5	0	0	0	4	1	0	1	0	1
Broccoli	0	0	0	0	196	3	0	0	0	0	0	0	0	0	1
Cabbage	0	0	1	3	5	184	0	0	1	0	0	0	5	0	1
Capsicum	1	0	0	0	0	0	197	0	0	0	2	0	0	0	0
Carrot	0	0	0	0	0	0	8	179	0	0	0	1	0	0	12
Cauliflower	0	0	0	1	7	7	0	0	174	0	1	0	9	0	1
Cucumber	10	2	5	9	0	1	0	0	1	171	0	0	1	0	0
Papaya	0	0	3	5	0	0	2	0	2	0	187	0	0	0	1
Potato	0	0	0	0	0	0	1	1	0	0	0	195	0	3	0
Pumpkin	2	0	0	0	9	1	0	0	1	0	0	0	185	0	2
Radish	0	0	0	0	0	0	0	0	7	0	0	3	0	190	0
Tomato	3	0	1	0	1	16	7	1	5	0	0	2	5	0	159

Image input	Output prediction (SoftMax)
	

Bibliografie

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

https://keras.io/api/data_loading/image/

https://machinelearningmastery.com/how-to-load-large-datasets-from-directories-for-deep-learning-with-keras/

https://www.tensorflow.org/guide/data

https://www.tensorflow.org/api_docs/python/tf/keras/layers/MaxPool2D

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

https://en.wikipedia.org/wiki/Activation_function

https://victorzhou.com/blog/softmax/

https://www.datacamp.com/tutorial/loss-function-in-machine-learning

https://www.geeksforgeeks.org/deep-learning/categorical-cross-entropy-in-multi-class-classification/

https://www.geeksforgeeks.org/machine-learning/metrics-for-machine-learning-model/

https://www.geeksforgeeks.org/javascript/tensorflow-js-tf-layersmodel-class-compile-method/

https://kambale.dev/build-compile-and-fit-models-with-tensorflow

https://www.geeksforgeeks.org/deep-learning/model-fit-in-tensorflow/

https://www.geeksforgeeks.org/deep-learning/model-evaluate-in-tensorflow/

https://www.v7labs.com/blog/confusion-matrix-guide

https://www.geeksforgeeks.org/machine-learning/confusion-matrix-machine-learning/