Mini Project Fundamentals of Soft Computing
VEGETABLE RECOGNIZER

Submitted By:

Priyanshu Sharma 9919103186 F-7

Piyush Tripathi 9919103132 F-7

Bhavya Varshney 9919103142 F-5

Naman Tyagi 9919103127 F-5

Submitted To:

Dr. Ashish Kumar

ABSTRACT

From vegetable production to delivery everything is done manually. Vegetable classifiers can automate the classification process and hence can help in improving packaging methods. Our aim is to train models which will be able to recognize different types of vegetables using Convolutional Neural Networks. Dataset contains 15 different types of vegetables.

As packaging and delivering industry is growing, companies wants to deliver their products faster than ever. This model can be used in mass packaging factories to identify and classify vegetables without any help of human and can help reducing packaging time for vegetables.

Tools and Technology Used

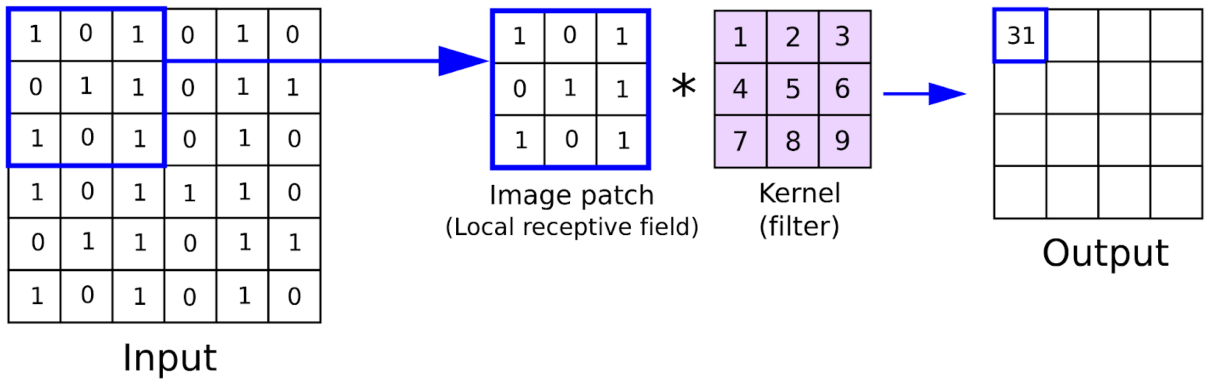
- Python
- Matplotlib
- Numpy
- Tensorflow
- Keras

THEORY

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

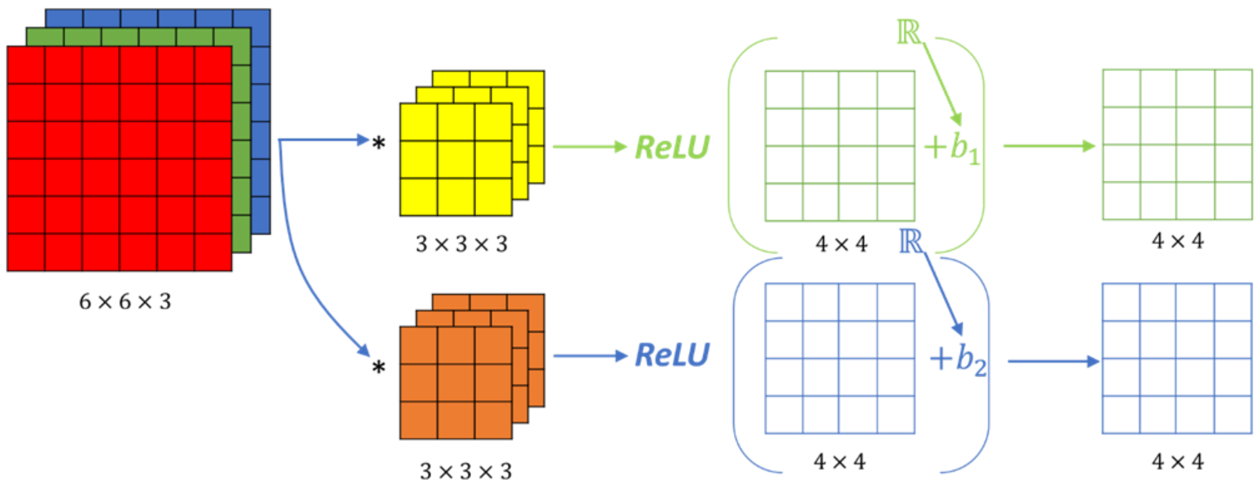
Convolution is a linear operation involving the multiplication of a set of weights with the input images represented by metrics similar to traditional neural networks. Here an array of weights is called a filter or kernel.



Strides defines the motion of the filter; if you set stride=1, which is the default value, the kernel takes one step at a time.

Usually, the filter size is smaller than the input data, and the type multiplication applied between filter and filter sized sample of input data is the dot product. A dot product is an element-wise multiplication between filter weights and filter sized sample of input data, summed up in a single value.

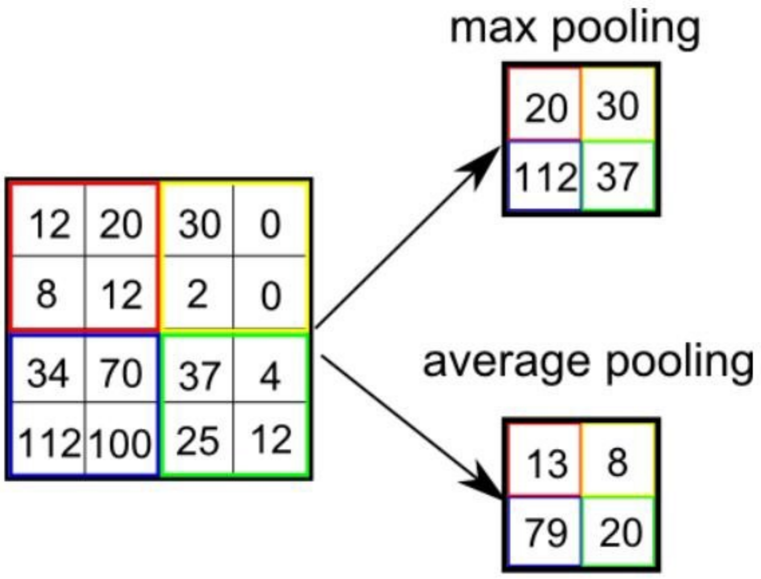
Intentionally, the filter size is chosen smaller than that of input data as it allows the same set of filter weights to be multiplied by the input array multiple times at different points on the image. In simple words, the filter is applied systematically to each filter sized input data from left to right and top to bottom.



This systematic application of the same filter throughout the same image is used to detect specific types of features in input data. As mentioned earlier, the output from the dot product of filter and input image for one time is a single scalar value. This filter is applied multiple times to the input image that results in a two-dimensional output array representing the filter of the input image. Such a two-dimensional output array is called a feature map, and this feature map then passed through some non-linearity like ReLU.

Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to decrease the computational power required to process the data through dimensionality reduction. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model.

There are two types of Pooling: Max Pooling and Average Pooling. Max Pooling returns the maximum value from the portion of the image covered by the Kernel. On the other hand, Average Pooling returns the average of all the values from the portion of the image covered by the Kernel.

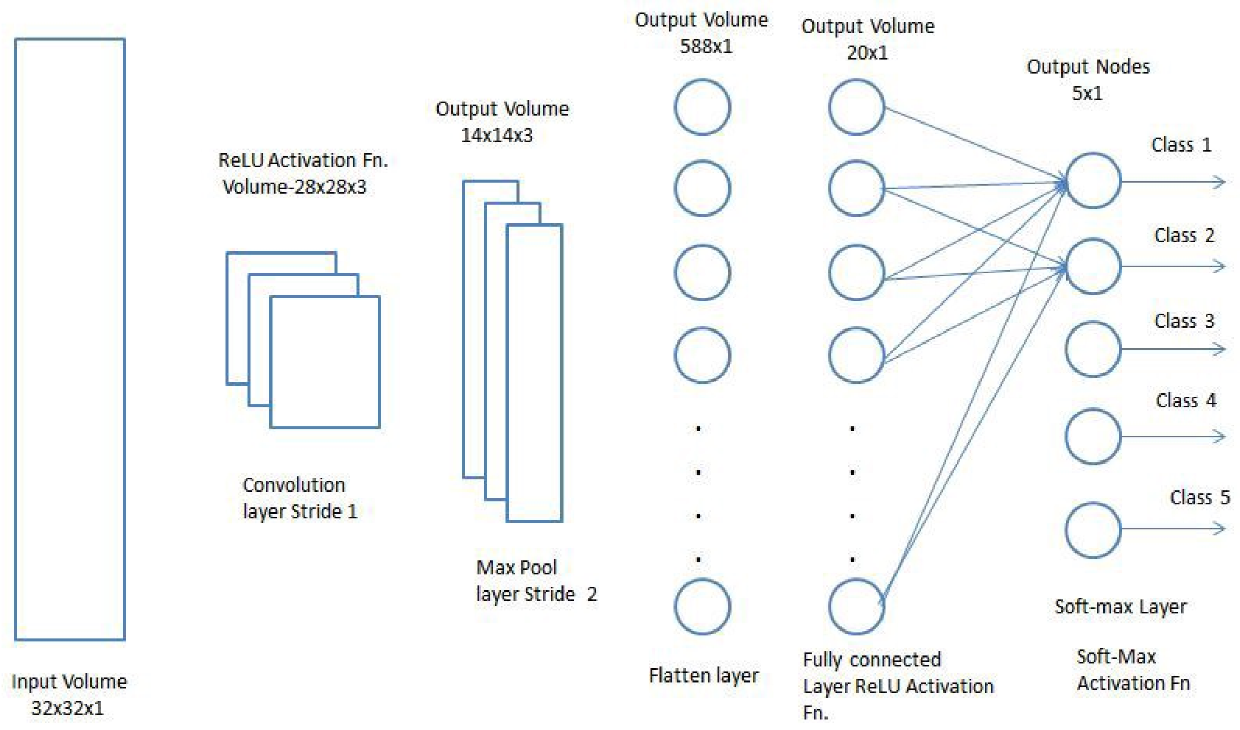


The Convolutional Layer and the Pooling Layer, together form the i-th layer of a Convolutional Neural Network. Depending on the complexities in the images, the number of such layers may be increased for capturing low-levels details even further, but at the cost of more computational power.

After going through the above process, we have successfully enabled the model to understand the features. Moving on, we are going to flatten the final output and feed it to a regular Neural Network for classification purposes.

Adding a Fully-Connected layer is a (usually) cheap way of learning non-linear combinations of the high-level features as represented by the output of the convolutional layer. The Fully-Connected layer is learning a possibly non-linear function in that space.

Now that we have converted our input image into a suitable form for our Multi-Level Perceptron, we shall flatten the image into a column vector. The flattened output is fed to a feed-forward neural network and backpropagation applied to every iteration of training. Over a series of epochs, the model is able to distinguish between dominating and certain low-level features in images and classify them using the Softmax Classification technique.



DATASET USED

Dataset Link: https://www.kaggle.com/datasets/misrakahmed/vegetable-image-dataset

The initial experiment is done with 15 types of common vegetables that are found throughout the world. The vegetables that are chosen for the experimentation are- bean, bitter gourd, bottle gourd, brinjal, broccoli, cabbage, capsicum, carrot, cauliflower, cucumber, papaya, potato, pumpkin, radish and tomato. A total of 21000 images from 15 classes are used where each class contains 1400 images of size 224×224 and in *.jpg format. The dataset split 70% for training, 15% for validation, and 15% for testing purpose.

This dataset contains three folders:

- train (15000 images)
- test (3000 images)
- validation (3000 images)

each of the above folders contains subfolders for different vegetables wherein the images for respective vegetables are present.

In this dataset there are 21000 images from 15 classes, where each class contains a total of 1400 images. Each class has an equal proportion and image resolution is 224×224 and in *.jpg format. We split our dataset into three parts, where 70%(approx.) for training and 15%(approx.) for testing, and the rest 15%(approx.) for validation.

IMPLEMENTATION

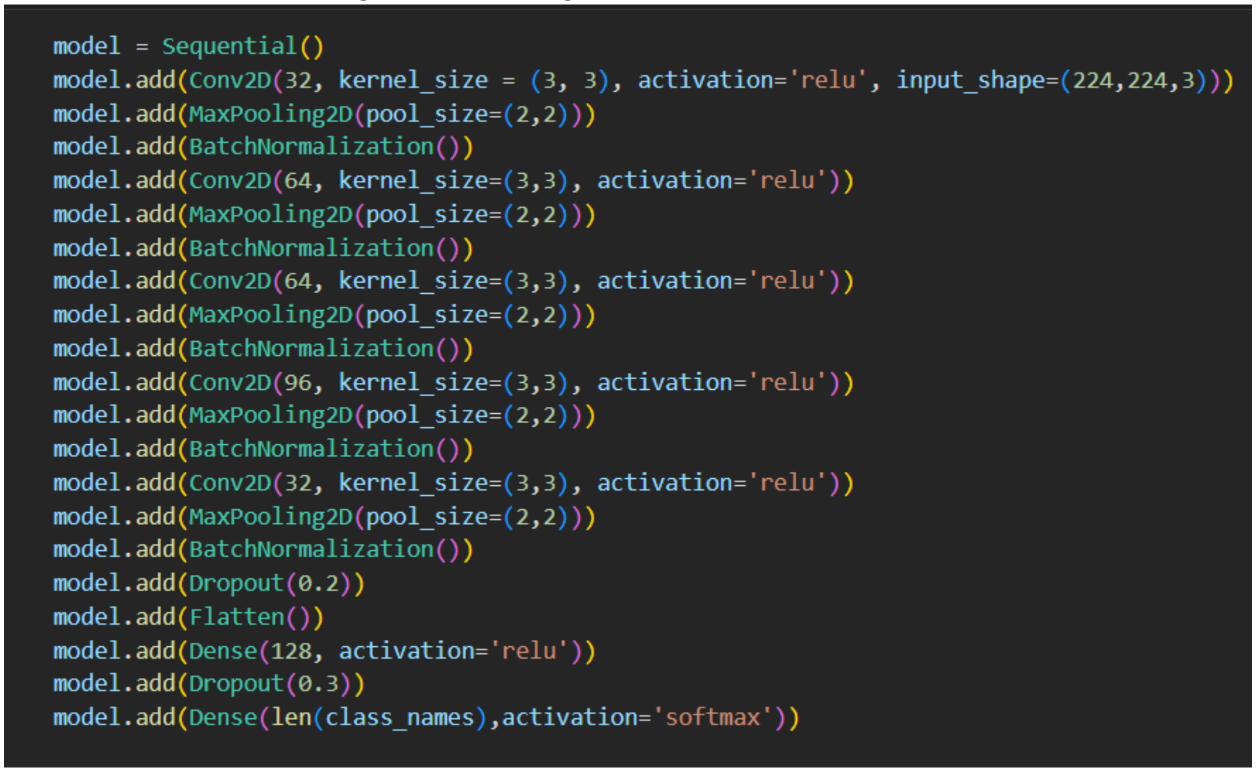
The dataset has different test, train and validation folders containing images of different vegetables. We read this dataset using keras preprocessing library. We use a sequential model from tensorflow which has 5 convolutional layers with 32, 64, 64, 96, 32 filters respectively and each with kernel size of 3X3 and activation function relu. After each convolutional layer there is a max pooling layer of 2X2 and a batch normalization layer.

Then we apply the dropout layer which randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting. Inputs not set to 0 are scaled up by 1/(1 - rate) such that the sum over all inputs is unchanged.

flattens the multi-dimensional input tensors into a single dimension, so you can model your input layer and build your neural network model, then pass those data into every single neuron of the model effectively.

Dense layer is the regular deeply connected neural network layer. It is the most common and frequently used layer. Dense layer does the below operation on the input and returns the output. output = activation(dot(input, kernel) + bias).

Compile defines the loss function:-The loss function is the function that computes the distance between the current output of the algorithm and the expected output.
 the optimizer:-An optimizer is a function or an algorithm that modifies the attributes of the neural network, such as weights and learning rate.



```
model = Sequential()
model.add(Conv2D(32, kernel_size = (3, 3), activation='relu', input_shape=(224,224,3)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Conv2D(96, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(len(class_names),activation='softmax'))
```

metrics:-They're used to train a machine learning model (using some kind of optimization like Gradient Descent), and they're usually differentiable in the model's parameters. Metrics are used to monitor and measure the performance of a model (during training and testing), and don't need to be differentiable

Model fitting is a measure of how well a machine learning model generalizes to similar data to that on which it was trained. A model that is well-fitted produces more accurate outcomes. A model that is overfitted matches the data too closely. A model that is underfitted doesn't match closely enough.input is train_dataset,20 epochs and validation is Val_dataset.

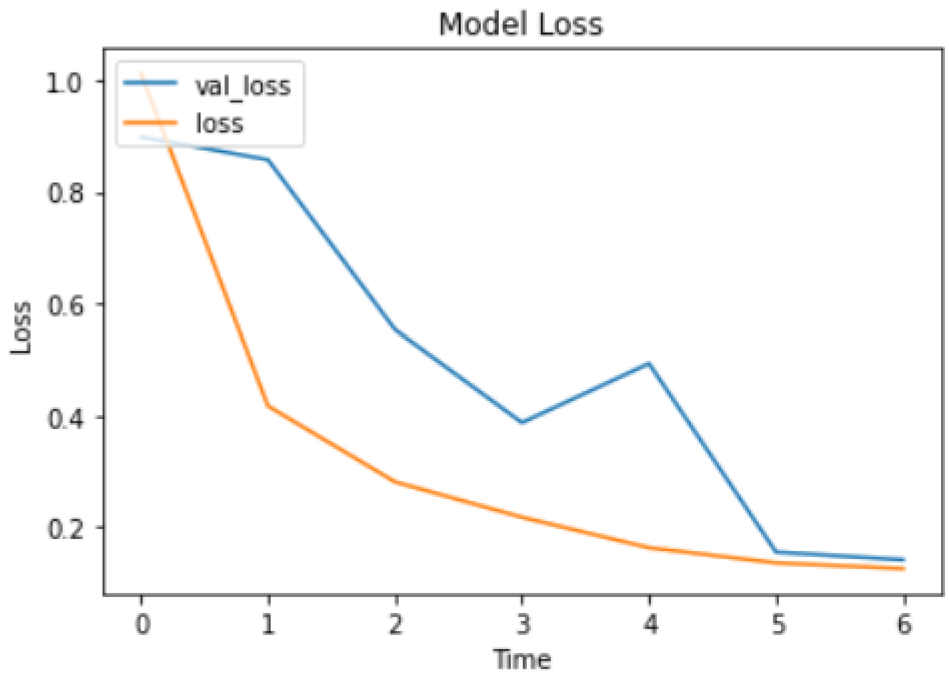
After that we have plotted Loss and accuracy which is implemented with the help of matplotlib library.

Finally, we are taking test data for testing the different vegetables and predicting them.

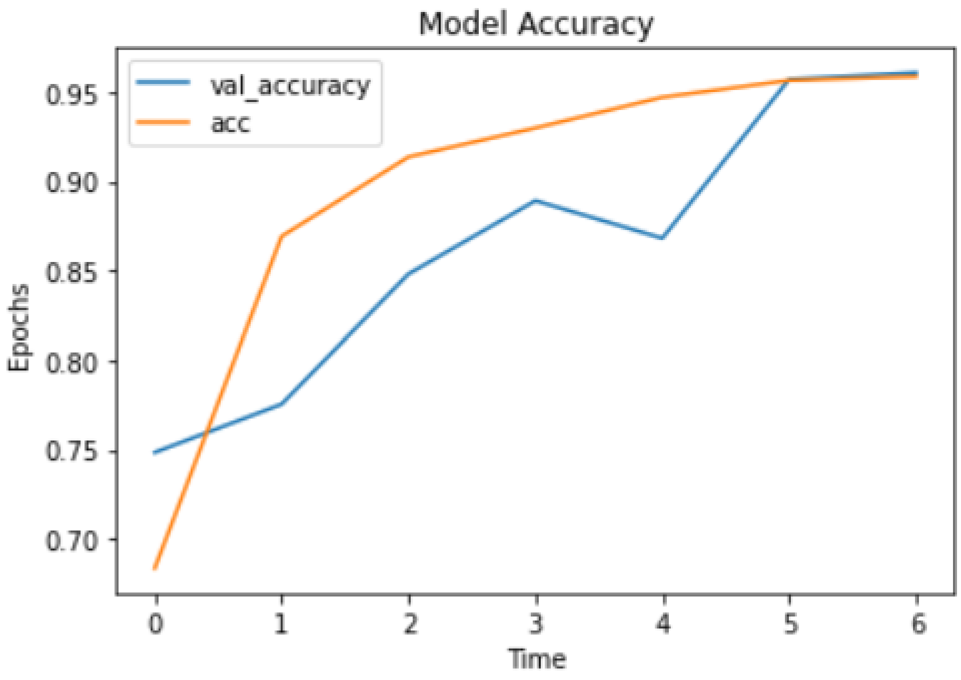
RESULT

Our CNN model successfully recognize and classify different vegetables. It provides 96.07% accuracy and have capability to recognize a vegetable having very different features like colour, size, texture etc.

LOSS:

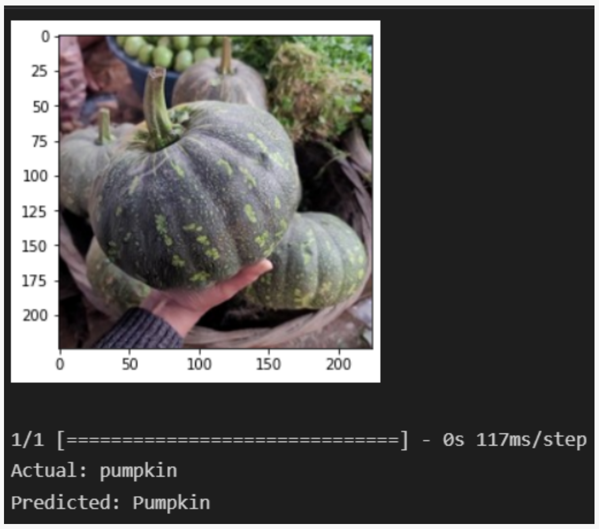


ACCURACY:

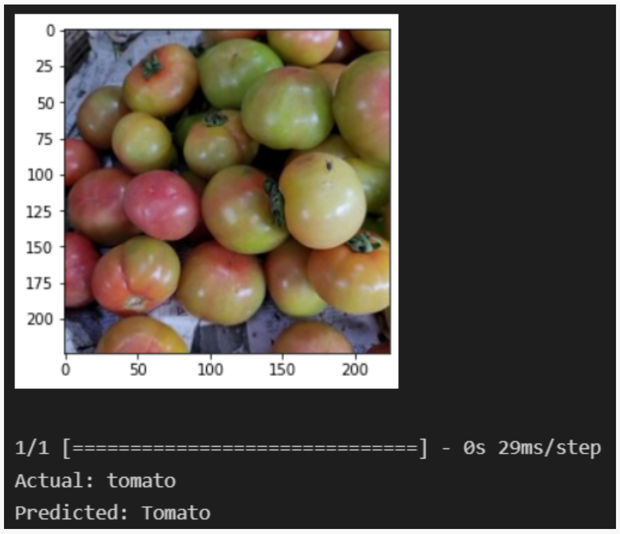


```
94/94 [==============================] - 21s 223ms/step - loss: 0.1408 - accuracy: 0.9607
```

PREDICTION:



```
1/1 [==============================] - 0s 117ms/step
Actual: pumpkin
Predicted: Pumpkin
```



```
1/1 [==============================] - 0s 29ms/step
Actual: tomato
Predicted: Tomato
```

REFERENCES

https://www.kaggle.com/datasets/misrakahmed/vegetable-image-dataset

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

https://www.tensorflow.org/api_docs

https://keras.io/

https://matplotlib.org/stable/index.html

https://numpy.org/doc/