Convolutional Neural Networks with Fruits Dataset

for

Machine Learning Project

by

Marticola No: 942091

Amanpreet singh

Università degli Studi di Milano

Declaration

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

TABLE OF CONTENTS

Chapter 1

Project Description

1.1 Need for Convolutional Neural Networks

Computer vision is one of the area in Machine Learning that's been advancing rapidly thanks to deep learning .Computer vision is now helping self-driving cars figure out where are the other cars and the pedestrians,making face recognition work much better than ever before .Computer vision problem is that the inputs can get really big for example in with 64 X 64 images we have to do 64 X 64 X 3 because there are three colour channels but that's not too bad but 64 X 64 is actually very small image if we work with larger images maybe a 1000 pixel X 1000 pixel image and that's actually just some one mega pixel but the dimension of the input features will be 1000 X1000X3 because we have three RGB channels and that's three million. So if we have three million input features then this means that X here would be three million dimensional and so if in the first hidden layer maybe we have just a thousand hidden units then the total number of weights that is in the matrix w1 if we use a standard fully connected Network,it will be a one thousand by three million dimensional matrix.This means that this matrix will have 3 billion parameters which is very large and with that many parameters, it is difficult to get enough data to prevent in network from Overfitting and also the computational requirements and the memory requirements to train the neural network with the billion parameters is infeasible.So here ,Convolution operation and Convolutional Neural Network comes in.

1.2 Convolutional Neural Networks -Edge Detection

The basic idea behind CNN's are adding filter matrix between input ,and "convoluting" ,the defined filter with input matrix of a photo,to get an output matrix.

Image Matrix-Image matrix is converting an image to numpy array matrix,which contains information of different colors/contours in each pixel according to RGB channel.

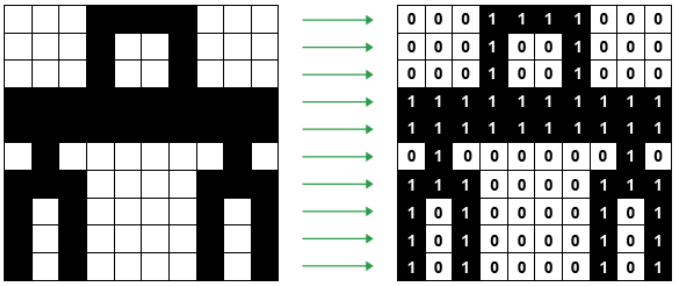


Figure 1.1: Image Matrix

Filters/Kernels- Filter/Kernals are matrix that are defined to find shapes and edges in an image.A filter or matrix can be 3x3 with defined range of colors values, and a filter can be used to find horizontal shapes by defining the selected color values of RGB horizontally or vertically to find vertical shapes and edges in an image which will give an output matrix ,with recorded numbers.

Figure 1.2: Color shading with horizontal and Vertical filters

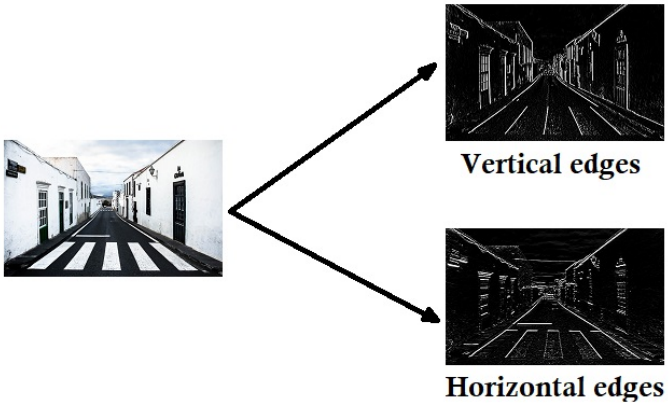


Figure 1.3: Edges detection

1.3 Padding

Padding is a process of adding additional boundaries to the pixel matrix, so during the convolutional process, the information on the corners is not lost, for example adding a boundary of 0's around the matrix. As in the image it is visible, without padding we would have used the corners with real information only once during convoluting, but now with padding , the original corners will be used more times than once.

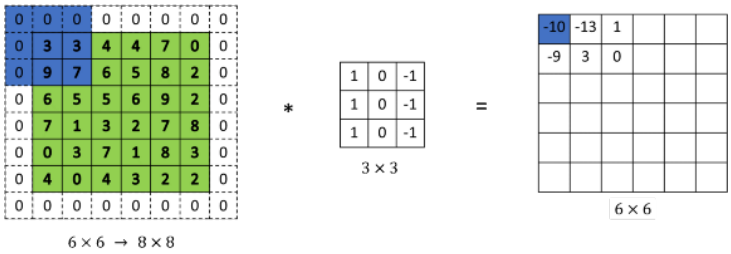


Figure 1.4: Padding

1.4 Pooling

Pooling is adding a layer after getting output from convoluting layer.What Pooling does is, simply reduce the dimensions of the matrix or in other words reduce the size of the output image from the convolutional output for a faster and more accurate process. To have basic intuition, of what happens in pooling is,we define a matrix for example 2x2 matrix, and it will divide the output matrix into 2x2 multiple matrices and will keep only points where it seems that there is some shape, or object recorded by figuring out the max value in particular 2x2 matrix and then this will further help in reducing the dimension of the output image for faster processing to go deeper into the network.

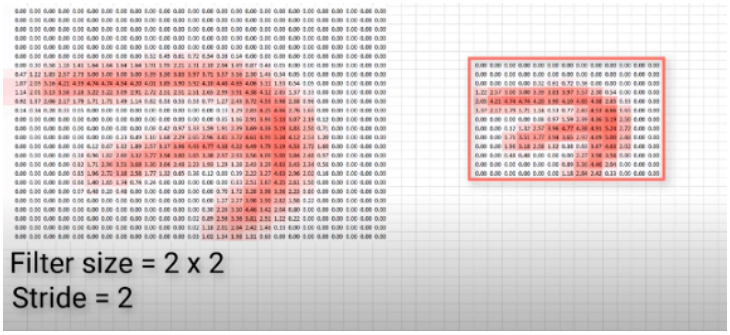


Figure 1.5: Pooling

1.5 Network Architectures

Network Architectures is architecture which transforms the image arrays to output martices while we add different layers like CNN or max pools or fully connected layers which helps in finding multiple features /shapes /images or so called **parameters** in an image as we go deep in the network.As we go deepen into the network,we see that width of an image array goes down,though number of channels are tend to increase. One can make its own Custom architecture which do not blow gradients or follow the Classic Neural Networks like

- RSNET
- VGG-13
- VGG-16
- ALEXNET

For this project I created

1.5.1 1 CUSTOM BASIC CNN ARCHITECTURE-

CNN→ $RELU \rightarrow MAXPOOL \rightarrow CNN \rightarrow RELU - MAXPOOL \rightarrow CNN \rightarrow RELU \rightarrow MAXPOOL \rightarrow FLATTEN \rightarrow FC \rightarrow FC \rightarrow OUTPUT$

1.5.2 2 VGG 13 Classic Neural Network

2 x CONV $\rightarrow RELU \rightarrow MAXPOOL \rightarrow 2xCONV \rightarrow RELU \rightarrow MAXPOOL \rightarrow 2xCONV \rightarrow RELU \rightarrow MAXPOOL \rightarrow 2xCONV \rightarrow RELU \rightarrow MAXPOOL \rightarrow 2xCONV \rightarrow RELU \rightarrow MAXPOOL \rightarrow FLATTEN \rightarrow FC1 \rightarrow FC2 \rightarrow FC3 \rightarrow$ *output*

Chapter 2

Project Algorithm

2.1 Project Code Steps

Complete Code can be found on →
https://github.com/Alexamannn/CNN_from_scratch/blob/main/cnn_Custom_vgg13.ipynb

1. Import Data from Kaggle into Google Colab enviroment and Inflating into Directory
2.
   - Defining Test and Train Folders path
   - Checking for image sample and image shape into numpy array
3. Defining Classes and total number of fruits and vegetables in data set
4. Creating a train and test generator which flows from all the images in the train and test data and record the image arrays with class names with categorical channels of each image plus zooms the images for better training and functioning of model

5. 2.1.1 Performing Basic CNN architecture

CNN→ $RELU \rightarrow MAXPOOL \rightarrow CNN \rightarrow RELU - MAXPOOL \rightarrow CNN \rightarrow RELU \rightarrow MAXPOOL \rightarrow FLATTEN \rightarrow FC \rightarrow FC \rightarrow OUTPUT$

- No Padding used because image corners have no information and all data is already centered and clean
- 32 steps in each of 100 epochs,
- Filters used = 1-32 ,3x3 matrix,2-32,3x3 matrix,3-64,3x3 matrix
- Max pooling with zero strides after every filter.

6.717,539 PARAMETERS with 98 % Accuracy over 131 Classes

6.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 98, 98, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 49, 49, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 47, 47, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 23, 23, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 21, 21, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 10, 10, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 6400)              0
_________________________________________________________________
dense (Dense)                (None, 1024)              6554624
_________________________________________________________________
dropout (Dropout)            (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 131)               134275
=================================================================
Total params: 6,717,539
Trainable params: 6,717,539
Non-trainable params: 0
_________________________________________________________________
```

Figure 2.1: Custom CNN output summary after each layer

2.1.2 VGG13 CLASSIC CNN ARCHITECTURE

2 x CONV $\rightarrow$ $RELU$ $\rightarrow$ $MAXPOOL$ $\rightarrow$ $2xCONV$ $\rightarrow$ $RELU$ $\rightarrow$ $MAXPOOL$ $\rightarrow$ $2xCONV$ $\rightarrow$ $RELU$ $\rightarrow$ $MAXPOOL$ $\rightarrow$ $2xCONV$ $\rightarrow$ $RELU$ $\rightarrow$ $MAXPOOL$ $\rightarrow$ $2xCONV$ $\rightarrow$ $RELU$ $\rightarrow$ $MAXPOOL$ $\rightarrow$ $FLATTEN$ $\rightarrow$ $FC1$ $\rightarrow$ $FC2$ $\rightarrow$ $FC3$ $\rightarrow$ *output*

8,294,851 PARAMETERS with 93 % Accuracy over 131 Classes

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 98, 98, 64)        1792
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 96, 96, 64)        36928
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 48, 48, 64)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 46, 46, 128)       73856
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 44, 44, 128)       147584
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 22, 22, 128)       0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 20, 20, 256)       295168
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 18, 18, 256)       590080
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 16, 16, 256)       590080
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 8, 8, 256)         0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 6, 6, 512)         1180160
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 4, 4, 512)         2359808
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 2, 2, 512)         2359808
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 1, 1, 512)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1024)              525312
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_3 (Dense)              (None, 131)               134275
=================================================================
Total params: 8,294,851
Trainable params: 8,294,851
Non-trainable params: 0
_________________________________________________________________
```

Figure 2.2: VGG CNN output summary after each layer

Chapter 3

Results

3.1 Custom Basic Architecture

3.1.1 Overfitting Check through Learning Curves

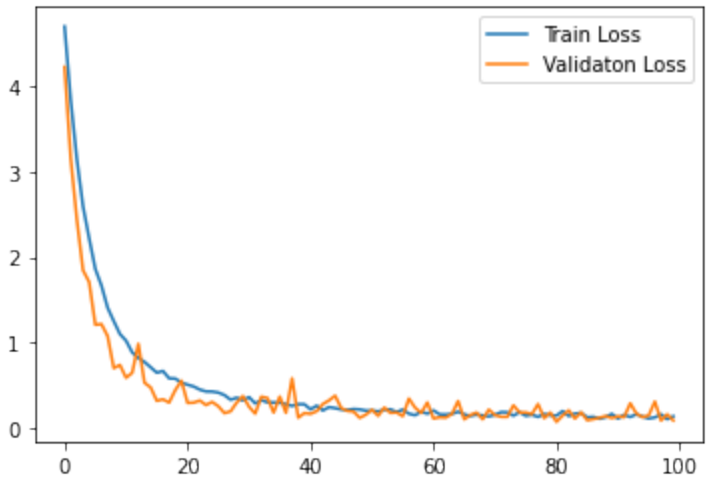


Figure 3.1: Train and Validation Learning Curves Showing a Good Fit

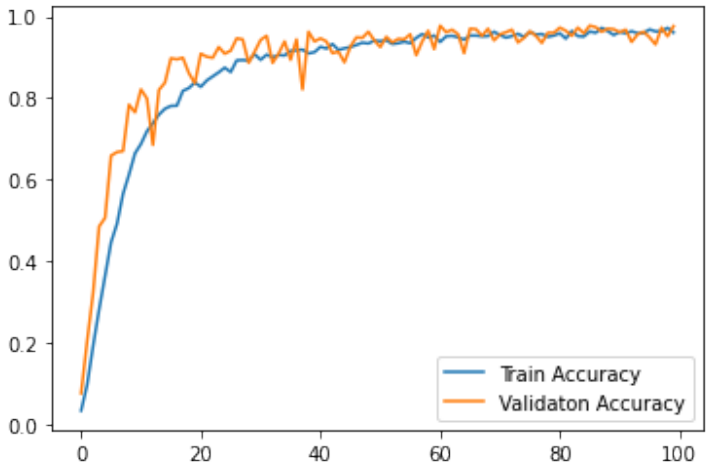
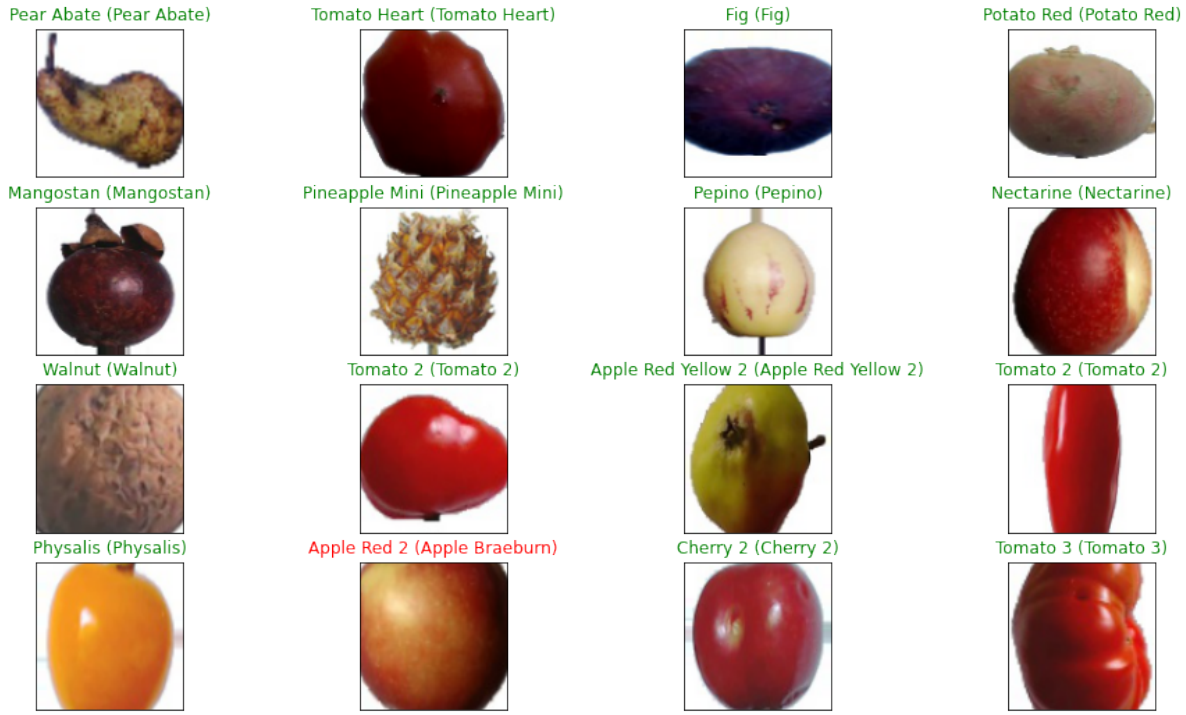
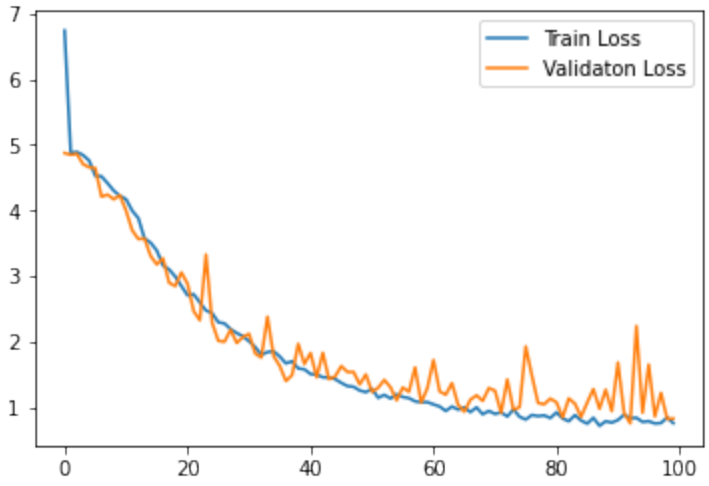


Figure 3.2: validation accuracy is parallel to training accuracy

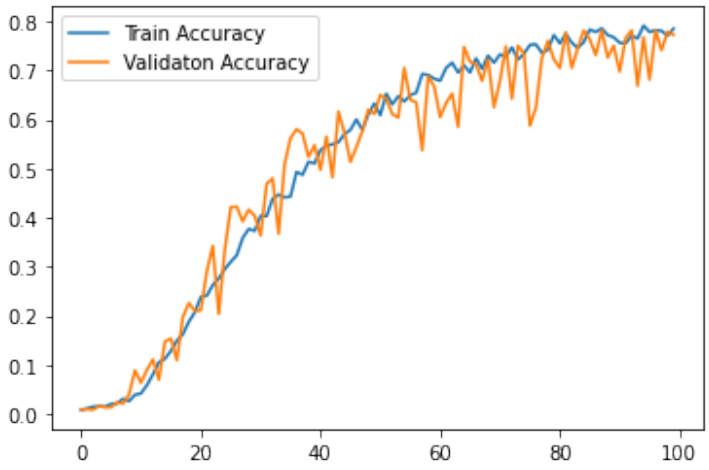
3.1.2 Predictions



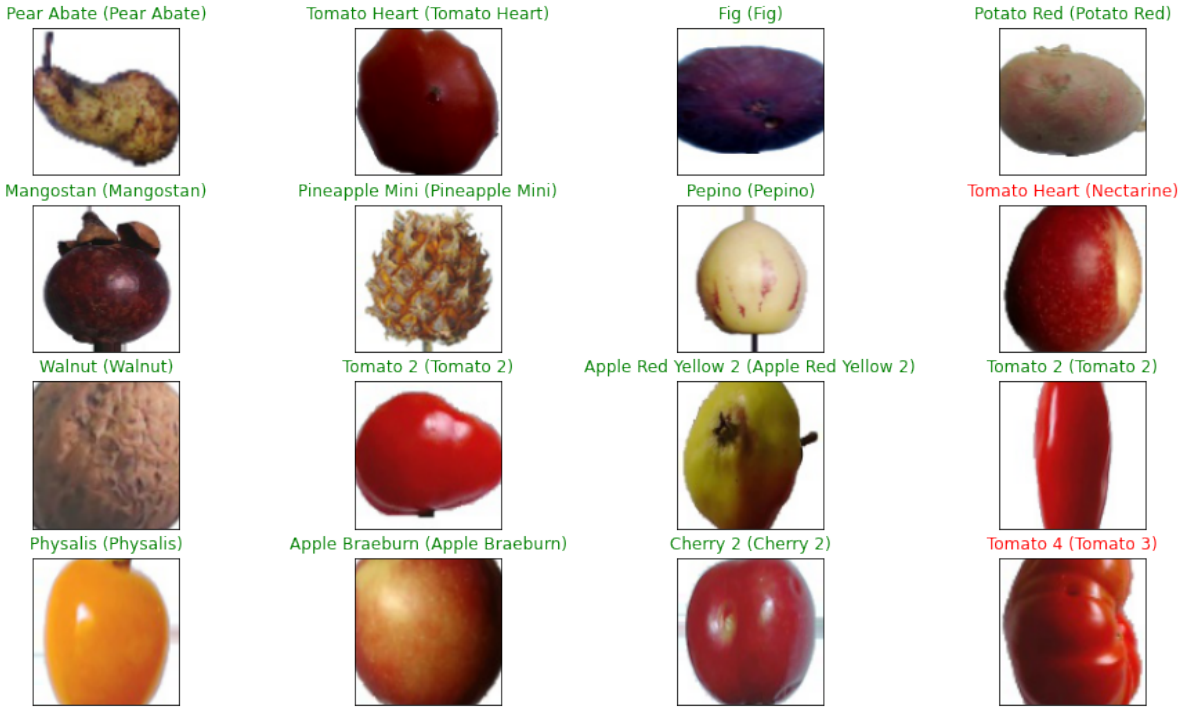
3.2 VGG13 Architecture



VGG13 Model shows fluctuations which means they had few noise and errors in the gradients during the process but they appear to vanish fast and model shows no over fitting



3.2.1 Predictions


Pear Abate (Pear Abate)
Tomato Heart (Tomato Heart)
Fig (Fig)
Potato Red (Potato Red)
Mangostan (Mangostan)
Pineapple Mini (Pineapple Mini)
Pepino (Pepino)
Tomato Heart (Nectarine)
Walnut (Walnut)
Tomato 2 (Tomato 2)
Apple Red Yellow 2 (Apple Red Yellow 2)
Tomato 2 (Tomato 2)
Physalis (Physalis)
Apple Braeburn (Apple Braeburn)
Cherry 2 (Cherry 2)
Tomato 4 (Tomato 3)