Fruit Recognition

Monis Khursheed - 2021UCS1711
Faraz Jawaid - 2021UCS1693
Saksham Raj - 2021UCS1713

Abstract

Fruit classification process is commercially important. Fruit production at harvest time is quite high. Classification of fruits according to their types and characteristics is usually done by hand and eye. This method can cause huge losses in terms of time, cost and labor. If we implement a ML model to classify fruits, we will save time and resources. We have built a fruit recognition model using tensorflow's sequential graph. We do image processing to improve the dataset so as to obtain optimum results. Compared to other machine learning (ML) algorithms like Support Vector Machines (SVM), deep neural networks (DNN) provide promising results to identify fruits in images.

Dataset

The dataset contains images of the following categories:

banana, apple, pear, grapes, orange, kiwi, watermelon, pomegranate, pineapple, mango, cucumber, carrot, capsicum, onion, potato, lemon, tomato, raddish, beetroot, cabbage, lettuce, spinach, soybean, cauliflower, bell pepper, chili pepper, turnip,

corn, sweetcorn, sweet potato, paprika, jalepeño, ginger, garlic, peas, eggplant.

The dataset contains three folders:

train (100 images each)
test (10 images each)
validation (10 images each)

Each of the above folders contains subfolders for different fruits wherein the images for respective food items are present.

Data Augmentation

We have also done some data augmentation to improve the testing accuracy of our model by reducing training bias. Each image is rotated, flipped horizontally and zoomed.

```
from tensorflow import keras
from tensorflow.keras import layers
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),
    ]
)
```

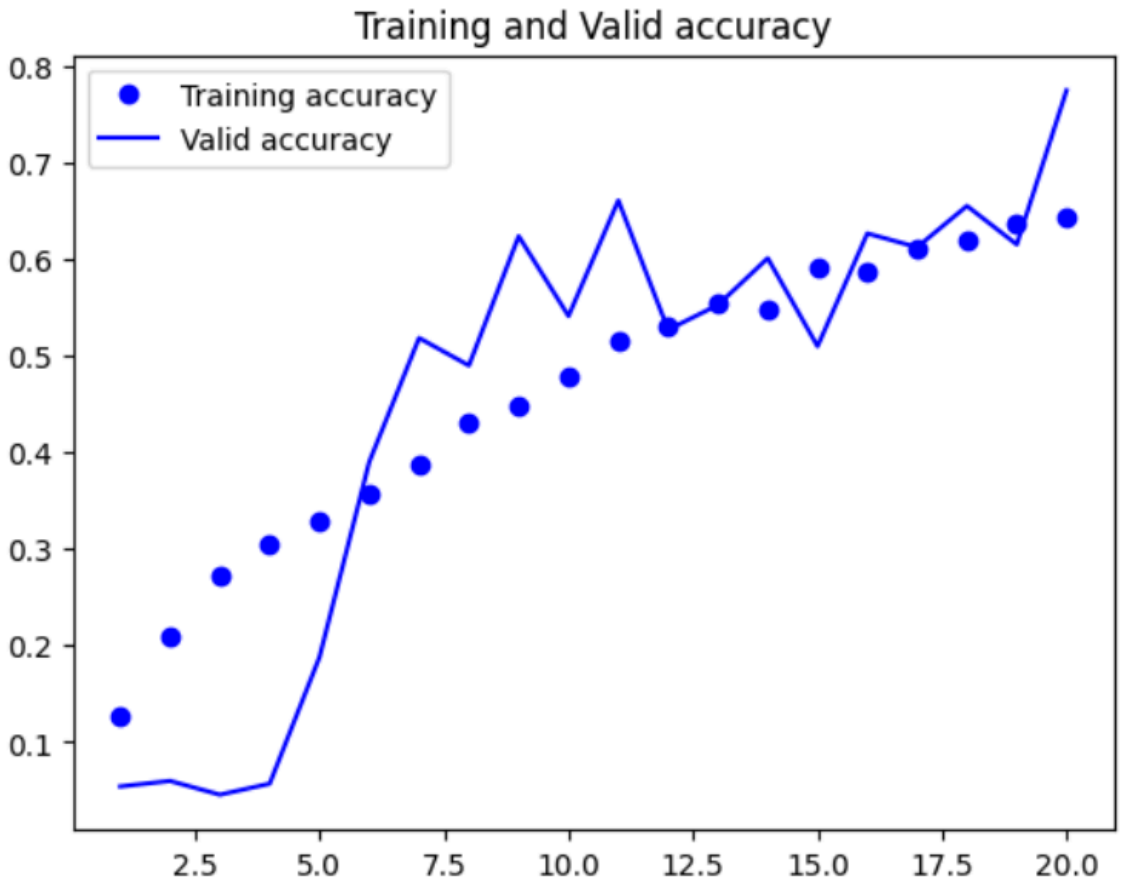
ML Model

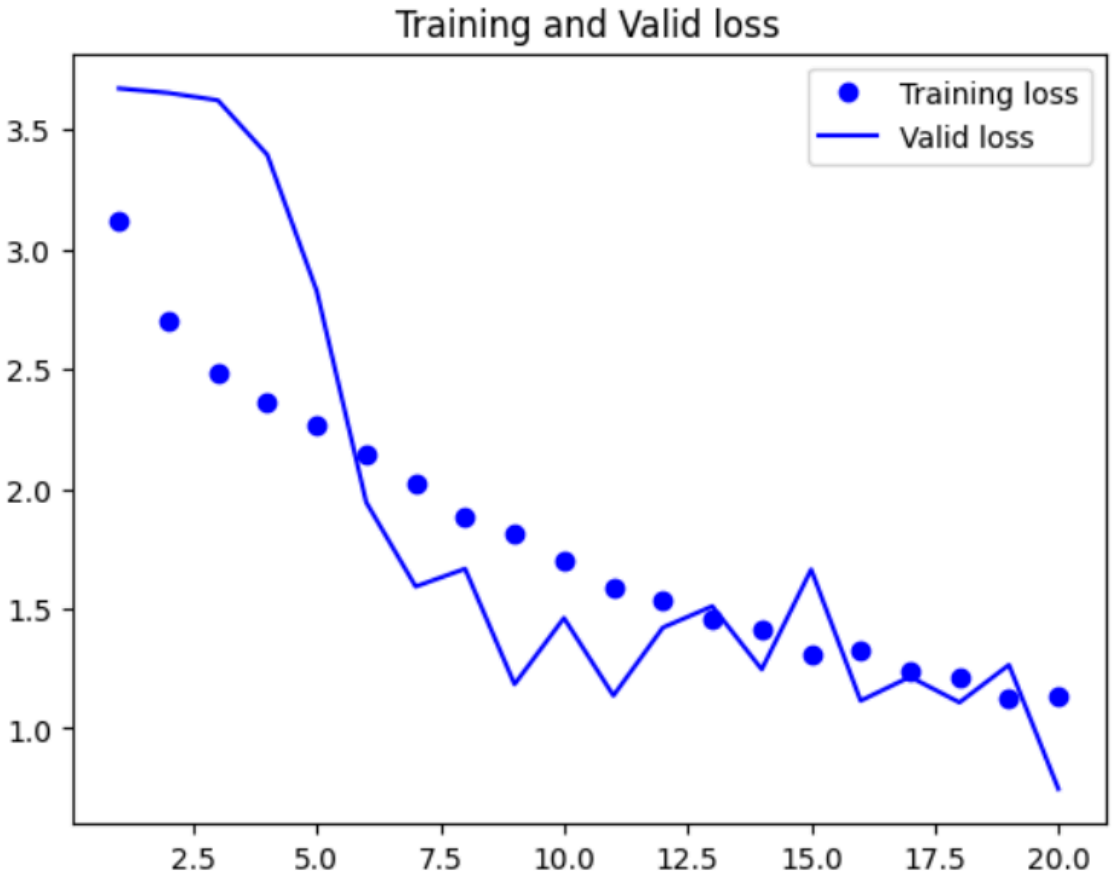
We have used the sequential CNN model using the keras wrapper of tensorflow. The model contains 32 filters of Conv2D layers with kernel size 2 and then two sets of separable Conv2D. The activation function is relu and we have also applied batch normalization and dropout after each layer. There are about 737k parameters, out of which 734k are trainable. We run this model for 20 epochs.

```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)

x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=2, use_bias=False)(x)

for size in [32, 64, 128, 256, 512]:
    residual = x
    x = layers.BatchNormalization()(x)
    x = layers.Activation("relu")(x)
    x = layers.SeparableConv2D(size, 3, padding="same", use_bias=False)(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation("relu")(x)
    x = layers.SeparableConv2D(size, 3, padding="same", use_bias=False)(x)
    x = layers.MaxPooling2D(3, strides=2, padding="same")(x)
    residual = layers.Conv2D(
        size, 1, strides=2, padding="same", use_bias=False)(residual)
    x = layers.add([x, residual])
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(36, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
model.summary()
```

Results

The final validation loss is **0.7472** and the final accuracy **77.49%**.


Training and Valid accuracy
Training accuracy
Valid accuracy
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Training and Valid loss
Training loss
Valid loss
3.5
3.0
2.5
2.0
1.5
1.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0