Fruits Classification

TA .Salma Elgyar

Team members

Name	ID
Yasmine Khaled atta	20201701154
Ahmed Esmail Mohamed	20201700024
Ebrahim Tarek Mohamed	20201700009
Ahmed Mohsen Fathelbab	20201700067
Ahmed Mohamed Kamal	20201700080

Table of contents

Introduction: Fruits Classification

Objective:

The goal of this image classification model is to accurately classify images of fruits into five distinct classes: apple, banana, grapes, mango, and strawberry contributes 1,980 images in each class. The model is trained to recognize and differentiate between these specific fruit categories.

Dataset:

The model is trained on a diverse dataset containing a large number of labeled images for each of the five fruit classes. The dataset is carefully curated to ensure a wide variety of fruit shapes, colors, and orientations are represented, allowing the model to generalize well to different real-world scenarios.

Data Preprocessing

1. **Label Encoding:**
    - The LabelEncoder is used to convert string class labels in the 'labels' list to numerical labels.
    - fit_transform method is applied to transform the class labels into numerical labels.
2. **Train-Test Split:**
    - train_test_split from is used to split the data into training and testing sets.
    - 80% of the data is used for training (train_size=0.80), and 20% is used for validation (validation_size=0.20).
    - random_state is set for reproducibility.
3. **Normalization:**
    - The pixel values of the images are normalized to the range [0, 1] by dividing by 255.0.

```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical

le = LabelEncoder()
train_labels = le.fit_transform(labels)
train_labels = to_categorical(train_labels, len(np.unique(train_labels)))

# split the data into training and testing
(trainX, testX, trainY, testY) = train_test_split(np.array(data), np.array(train_labels), test_size=0.20, random_state=42)

# normalize the data
trainX = trainX.astype("float") / 255.0
testX = testX.astype("float") / 255.0

print(len(trainX))
print(len(testX))
```

```
7920
1980
```

4. Data augmentation

We applied data augmentation to increase the size of the training dataset by applying various transformations to the original images such as transformations, including rotation, width and height shifts, shear, zoom, and horizontal flip.

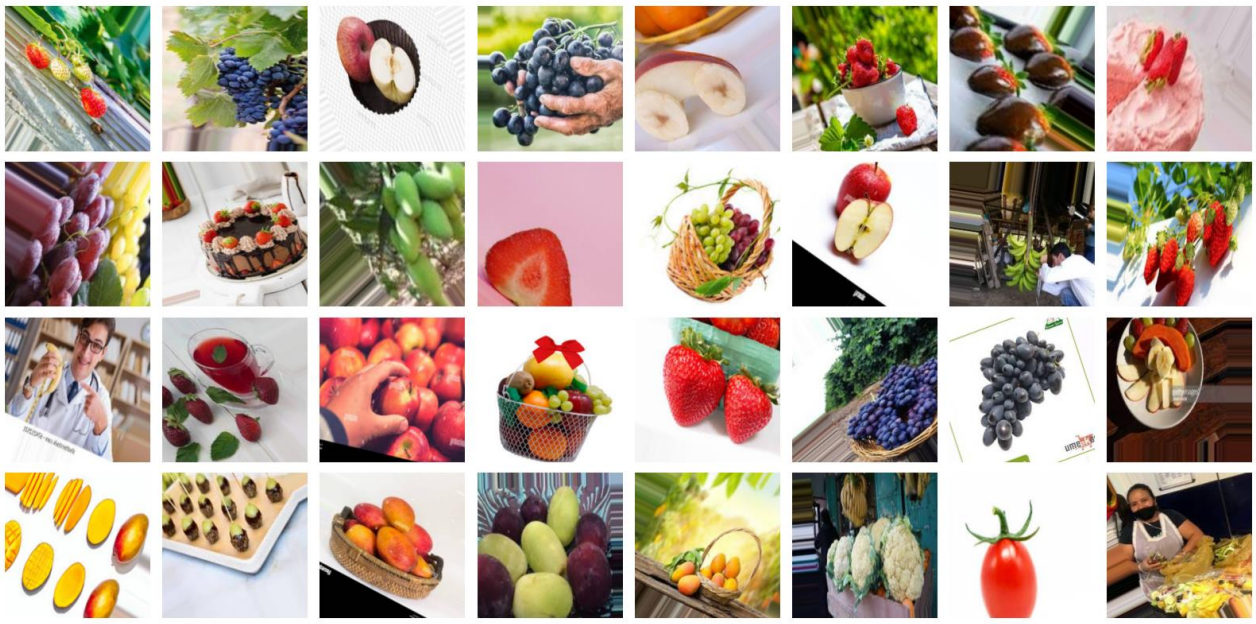
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
                                    rotation_range=40,
                                    width_shift_range=0.2,
                                    height_shift_range=0.2,
                                    shear_range=0.2,
                                    zoom_range=0.2,
                                    horizontal_flip=True)
train_gen = train_datagen.flow(trainX,trainY,batch_size=32)

print(len(train_gen))
```

```
248
```

- **Plotting randomly selected augmented images**



Data Exploration

- **Print the unique labels and their counts :**

```
unique_labels, label_counts = np.unique(labels, return_counts=True)
print(unique_labels)
print(label_counts)
```

```
['1' '2' '3' '4' '5']
[1980 1980 1980 1980 1980]
```

- **Plotting randomly selected images**

Model Engineering

- ## Alexnet Model Implementation

AlexNet Model Architecture:

The AlexNet model is defined as a Sequential model.

- Layer 1: Convolutional layer with 96 filters, kernel size of (11, 11), and a stride of (4, 4). Batch normalization and ReLU activation follow, and then a max-pooling layer with a pool size of (3, 3) and stride of (2, 2).
- Layer 2: Convolutional layer with 256 filters, kernel size of (5, 5), and padding. Batch normalization, ReLU activation, and max-pooling follow.
- Layer 3: Convolutional layer with 384 filters, kernel size of (3, 3), and padding. Batch normalization and ReLU activation follow.
- Layer 4: Another convolutional layer with 384 filters, kernel size of (3, 3), and padding. Batch normalization and ReLU activation follow.
- Layer 5: Convolutional layer with 256 filters, kernel size of (3, 3), and padding. Batch normalization, ReLU activation, and max-pooling follow.
- Flatten and Fully Connected Layers:
  - The model flattens the output and connects it to a dense layer with 4096 units. Batch normalization, ReLU activation, and dropout are applied.
  - Another dense layer with 4096 units follows, with similar batch normalization, ReLU activation, and dropout.
  - The final dense layer has 5 units (assuming it's a 5-class classification problem) with a softmax activation function.

AlexNet Model Accuracy:

A validation accuracy of 69% indicates that the model is correctly classifying about 69% of the examples in the validation set.

- # Inception Model Implementation

Inception Model Architecture:

- Input Layer: The model takes an input image of shape (IMG_SIZE, IMG_SIZE, 3) (assuming RGB images).
- Inception-like Layers (Layer 1, Layer 2, Layer 3):
  - Each Inception-like layer consists of multiple convolutional layers with different filter sizes applied in parallel.
  - The goal is to capture features at different scales.
  - Convolutional layers with kernel sizes of (1,1), (1,3), and (3,1) are used to process the input in different ways.
  - In Layer 2, additional convolutional layers are added for more complexity.
  - Layer 3 uses max-pooling followed by a 1x1 convolution and dropout.
- Concatenation (mid_1):

  The outputs of the Inception-like layers are concatenated along the depth (axis=3) to create a single tensor (mid_1).

- Flatten Layer:
  - The concatenated tensor is flattened to prepare for fully connected layers.
- Fully Connected Layers (dense_1, dense_2, dense_3):
  - Dense layers are used for higher-level feature extraction and abstraction.
  - The number of neurons decreases towards the output layer.

- Output Layer:

The final output layer has 5 neurons with a softmax activation function for multi-class classification.

Inception Model Accuracy:

A validation accuracy of 68% indicates that the model is correctly classifying about 68% of the examples in the validation set.

- # Mobilenet Model Implementation

- **Define Depthwise Separable Convolution Function**
  - A function is defined to create a depthwise separable convolution block, which is a type of convolutional layer often used in lightweight neural network architectures like MobileNet.
  - This block consists of two consecutive layers: a depthwise convolution layer and a pointwise convolution layer.
  - Batch normalization and ReLU activation functions are applied after each convolution operation.

- **Model Definition**
  - An input layer is created for images with a shape of (224, 224, 3), representing height, width, and color channels.
  - The initial layers include a standard convolutional layer followed by batch normalization and ReLU activation. These layers serve as the starting point for the neural network.

- **Loading Pre-trained Weights**
  - Pre-trained weights from a MobileNetV1 model are loaded. These weights capture features learned from a large dataset and can be beneficial for transfer learning on a new task.

- **Freezing Layers**
  - To preserve the pre-trained features, all layers in the model are set to non-trainable. This prevents the weights from being updated during the training process.

- **Fully Connected (FC) Head**
  - Flattening the output from the pre-trained layers to a 1D vector.
  - A dense layer with 256 units and ReLU activation function.
  - Batch normalization to normalize the output of the dense layer.
  - Dropout layer to prevent overfitting.
  - Final dense layer with softmax activation function for the 5 classes.

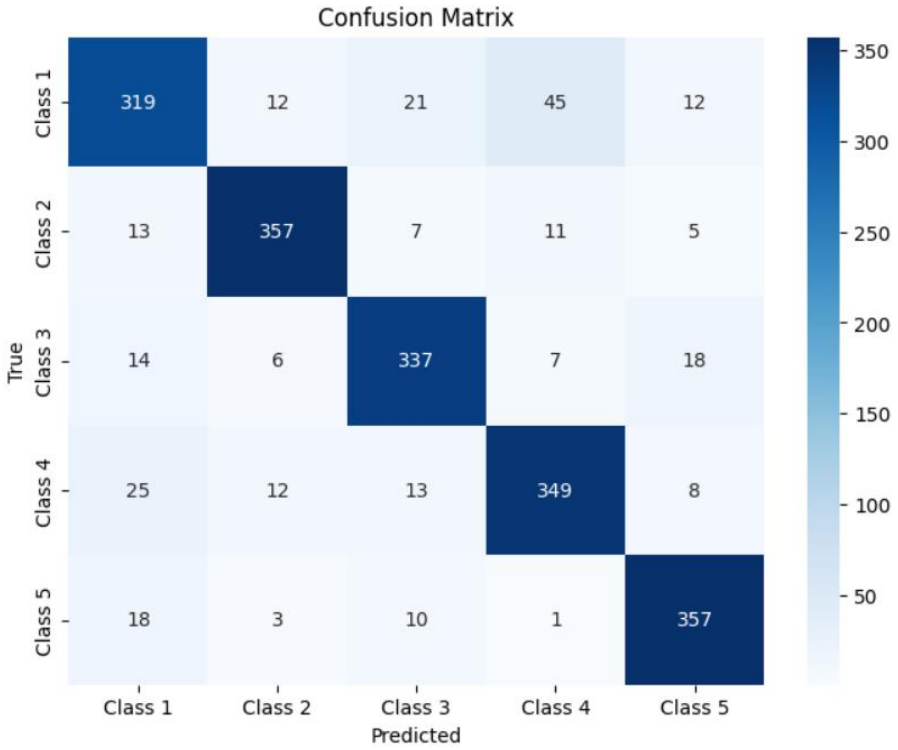
- ## Mobilenet Model Accuracy:

  A validation accuracy of 87% indicates that the model is correctly classifying about 87% of the examples in the validation set.
  A test accuracy of 89.9% indicates that the model is correctly classifying about 89.9% of the examples in the test set.

Best Model Evaluation

Mobilenet Model Evaluation:

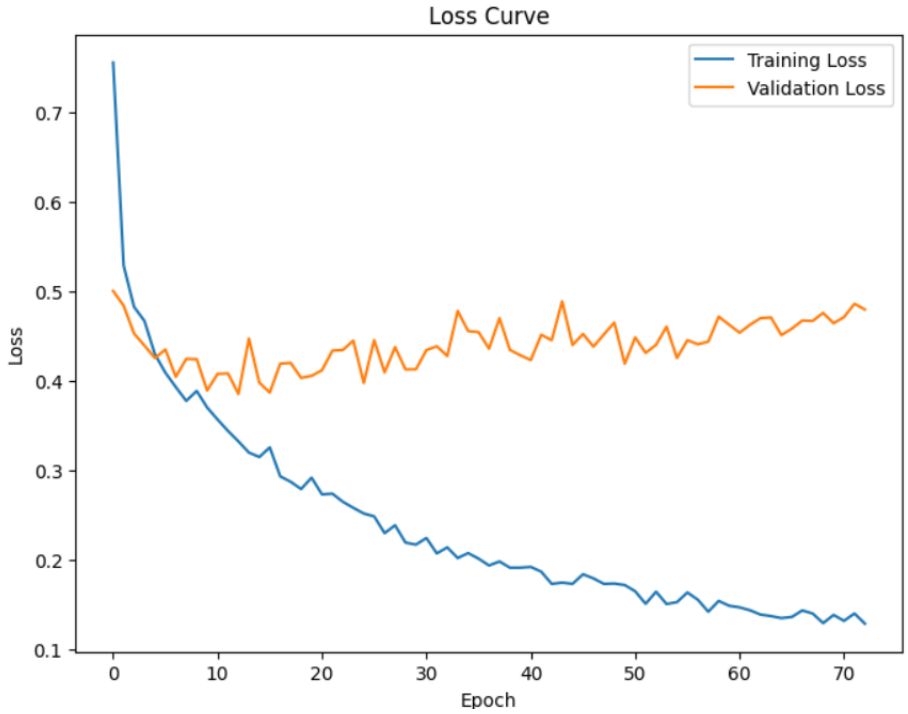
- **Confusion matrix**



- **Classification report**

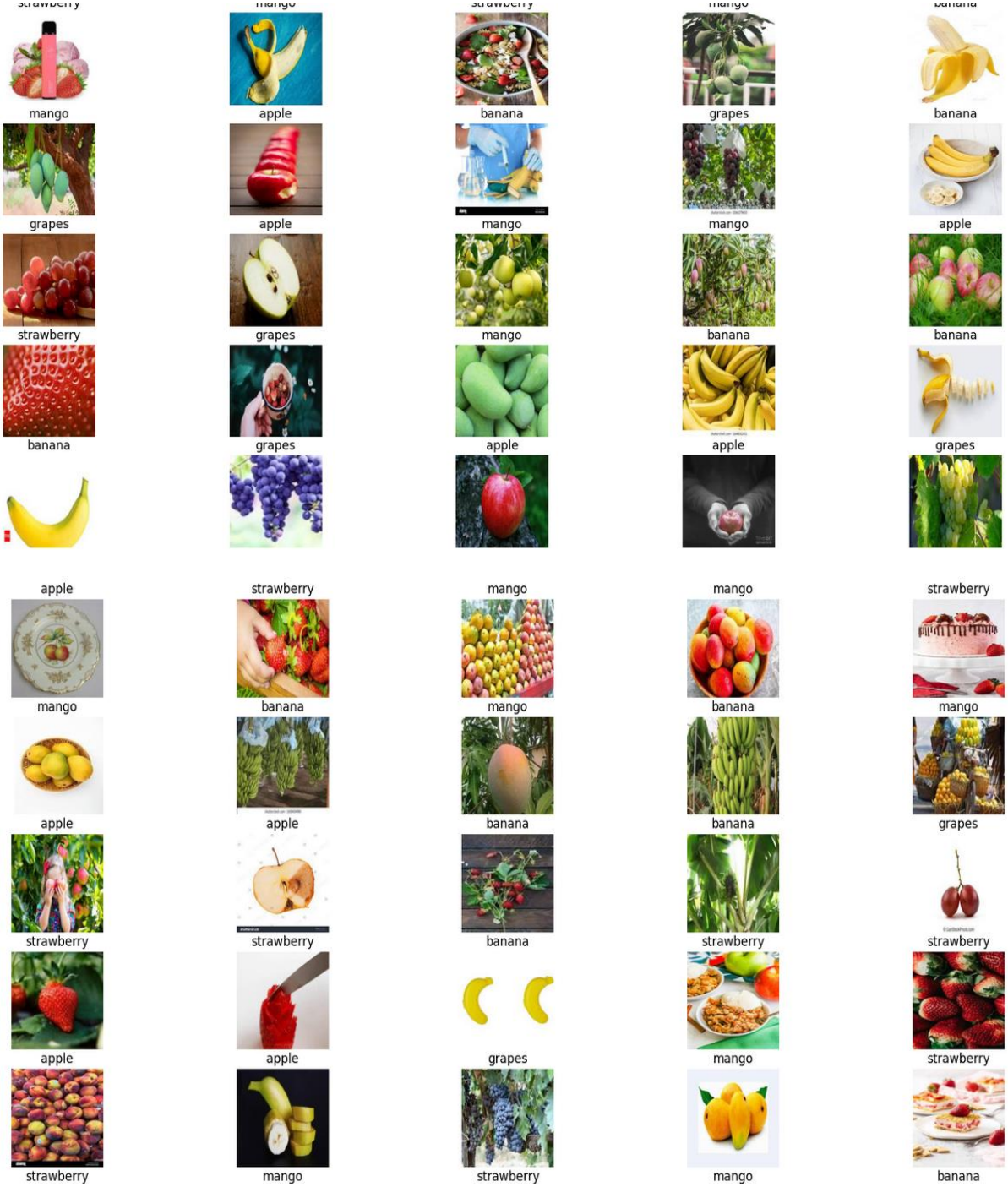
	precision	recall	f1-score	support
Class 1	0.82	0.78	0.80	409
Class 2	0.92	0.91	0.91	393
Class 3	0.87	0.88	0.88	382
Class 4	0.85	0.86	0.85	407
Class 5	0.89	0.92	0.90	389
accuracy			0.87	1980
macro avg	0.87	0.87	0.87	1980
weighted avg	0.87	0.87	0.87	1980

- **Loss Learning Curve**



- **Accuracy Learning Curve**

- **Visualize some of test images with predicted fruit names**



- ## Vision Transformer Implementation

Patch Extraction:

- The input image is divided into smaller non-overlapping patches.
- Each patch is treated as a separate unit for processing.
- This step is crucial for enabling self-attention mechanisms across different regions of the image.

Patch Encoding:

- Each patch is embedded using a learned projection (linear transformation) and combined with a special token known as the "class token."
- Positional information is added to the patch embeddings using positional embeddings.
- The class token helps the model capture global information from the entire image.

Transformer Blocks:

- The core of the Vision Transformer is the Transformer block.
- Each block consists of:
    - Layer normalization for stabilizing the input.
    - Multi-Head Self-Attention mechanism, which allows the model to focus on different parts of the input sequence.
    - Feedforward neural network (MLP) applied to the output of the attention mechanism.
    - Residual connections (skip connections) around both the attention mechanism and the MLP.
- This architecture enables the model to capture complex hierarchical patterns and long-range dependencies in the image.

Transformer Encoder:

- Stacks multiple Transformer blocks sequentially to form the complete Transformer Encoder.
- Each block processes the input sequentially, building a hierarchy of features.

Global Representation:

- After passing through the Transformer Encoder, the output is a set of encoded patches.
- A global average pooling operation is applied to obtain a fixed-size representation of the entire image.
- This global representation captures high-level features from the input image.

Multi-Layer Perceptron (MLP):

- The global representation is fed into an MLP, consisting of fully connected layers with activation functions.
- Dropout is applied to prevent overfitting.
- The final output of the MLP is used for image classification.

Model Architecture:

- The entire model is created by connecting the Patch Extraction, Patch Encoding, Transformer Encoder, and MLP layers.
- The model takes an input image and produces class predictions.

Vision Transformer Model Accuracy:

- A validation accuracy of 20% indicates that the model is correctly classifying about 20% of the examples in the validation set.

Conclusion

In conclusion, our project aimed to classify five classes of fruits using four models: AlexNet, Inception, MobileNet, and Vision Transformer. After thorough evaluation, the MobileNet model with pre-trained weights emerged as the top performer, achieving an impressive 89.9% accuracy on test samples.