**Fruit classification using Deep Learning**

We used Fruits 360 dataset, from kaggle which consists of 82,213 images of 120 different fruit and vegetable classes. we will build a model with Keras that can classify between 10 different types of fruit by using convolutional neural networks (ConvNets). To do that we followed the following steps, the following steps:

**PART I**

**Collect and label a dataset of fruit images.**

**Preprocess the images**. including resizing the images to a uniform size, converting them to a suitable format (e.g NifTi), and scaling the pixel values.

**Split the dataset into training and test sets**. We used 70% of the data for training, 15% for validation, and 15% for test.

**Define the model architecture.** to decide on the number and size of the convolutional filters, the size of the pooling windows, and the number and size of the dense layers.

**Compile the model**

Train the model. feeding the training data to the model and updating the model weights to minimize the loss function.

**Evaluate the model**. evaluate the performance of the model and determine how well it generalizes to unseen data.

**improve the model's performance**.

**PART II**

The dataset is split into a training set of 61,488 images and a test set of 20,622 images, with a train/test split of approximately 75/25. The images are 100 x 100 pixels in size and are in RGB format. The Project will walk I through the process of preparing the dataset, building the model, and training and evaluating the model. We will also learn how to visualize the results and how to fine-tune the model for better performance.

Using Convolutional neural networks (ConvNets) we are able build our fruit classification model using ConvNets. We will go through the entire process of building and training the model, including preparing the dataset and evaluating the model's performance. We will also discuss how to visualize the results and fine-tune the model for improved accuracy. If the goal is to recognize a wide variety of fruit, then a model that is specialized in recognizing a wide variety of fruit would likely be better. This model would have more training data and be more likely to accurately recognize a greater number of fruit classes.

On the other hand, if the goal is to recognize a smaller number of fruit classes with a high level of accuracy, then a more general model that can recognize a smaller number of fruit classes may be better. This model would likely have more focused training data and be able to achieve a higher level of accuracy on the specific classes it was trained on.

**Our goal is to recognize a wide range of fruits.**

A picture containing table, food, indoor, plate

Description automatically generated

Before we begin building our model, it is helpful to have a basic understanding of deep learning and ConvNets. We will provide a brief overview of these concepts and how they relate to image classification tasks. We will also show, how to visualize the Fruits 360 dataset and prepare it for deep learning.

Each layer of a CNN is made up of a set of filters that are applied to the input data to extract different features. The output of these filters is then passed through a non-linear activation function, which allows the network to learn more complex patterns in the data. The output of the activation function is then typically downsampled by a process called max pooling, which reduces the dimensionality of the data and helps the network to be more robust to small translations in the input.

**Visualization code**

**# Imports**

**import os**

**import matplotlib.pyplot as plt**

**from PIL import Image**

**# Configuration**

**dir\_with\_examples = './imagecopies'**

**files\_per\_row = 3**

**# List the directory and perform computations**

**files\_in\_dir = os.listdir(dir\_with\_examples)**

**number\_of\_cols = files\_per\_row**

**number\_of\_rows = math.ceil(len(files\_in\_dir) / number\_of\_cols) # use math.ceil to round up to the nearest integer**

**# Generate the subplots**

**fig, axs = plt.subplots(number\_of\_rows, number\_of\_cols)**

**fig.set\_size\_inches(8, 5, forward=True)**

**# Map each file to subplot**

**for i, file\_name in enumerate(files\_in\_dir):**

**image = Image.open(f'{dir\_with\_examples}/{file\_name}')**

**row = i // files\_per\_row # use integer division to compute the row index**

**col = i % files\_per\_row # use modulo to compute the column index**

**axs[row, col].imshow(image)**

**axs[row, col].axis('off')**

**# Show the plot**

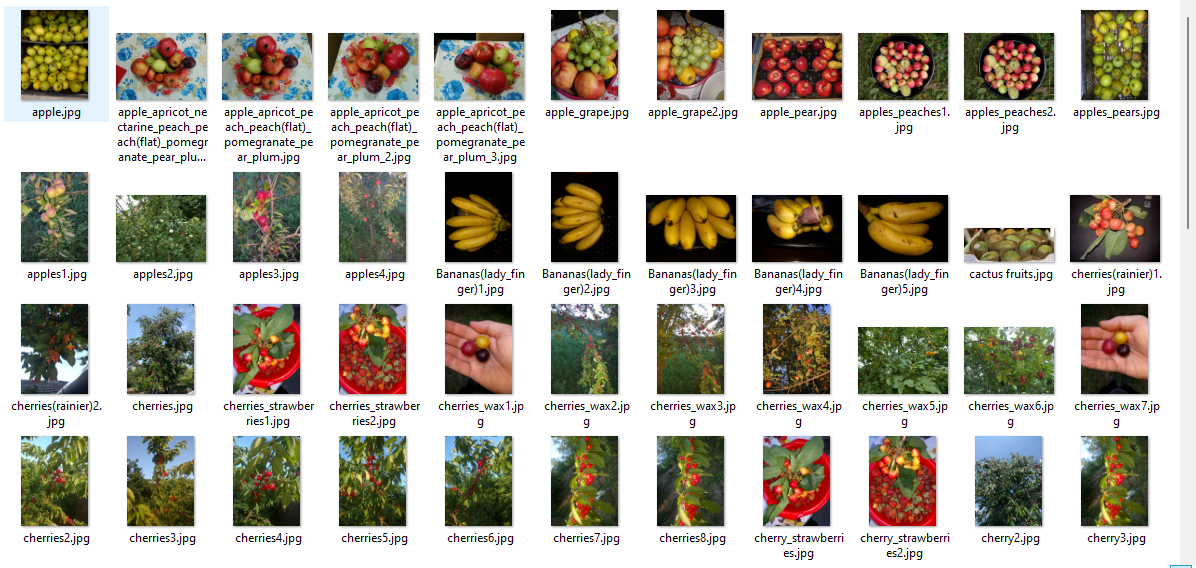
**plt.show()**

* Importing the math module to use the ceil function
* Using integer division and modulo to compute the row and column indices for the subplot, respectively
* Using a for loop to iterate over the elements in files\_in\_dir and their indices (using the enumerate function)
* Using the with statement to open the image file, which ensures that the file is closed properly when the block of code is finished executing

The output of the convolutional and pooling layers is then passed through one or more fully-connected (dense) layers, which perform the final classification task. The weights of the filters and dense layers are adjusted during training using an optimization algorithm, such as stochastic gradient descent, to minimize a loss function that measures the difference between the predicted and true labels of the training data.

Once trained, a CNN can be used to classify new images by forward propagating the image through the network and outputting the predicted label.

Fruit classification is a task in which a machine learning model is trained to recognize and differentiate between different types of fruits. In this article, we will explore how to use deep learning techniques and the Keras framework to build a fruit classifier.



To simplify things, we will manually select a few fruit classes to classify, rather than trying to create a model that can distinguish between all 120 types of fruit in the dataset. We will create two new folders, Training\_smaller and Test\_smaller, and copy the fruit classes we want to classify into these folders.

A picture containing background pattern

Description automatically generated

Next, we will use the Keras framework to build our model. We will start by importing the necessary dependencies and configuring our data and model parameters. Our model will consist of several layers, including convolutional layers and fully connected layers. We will also use an image data generator to load the images from our dataset.

Once our model is set up, we will compile it and begin training. We will train our model for 25 epochs, using the Adam optimizer and sparse categorical crossentropy as our loss function. After training, we will evaluate the performance of our model on the test set.

we are setting up some configuration options for a machine learning model. There are two main parts to the configuration: data configuration and model configuration.The data configuration consists of the paths to the training data and the testing data. These paths tell the model where to find the data it will use to learn and evaluate its performance.

**PART III**

The model configuration is a little more complex. It includes several different options that control how the model is trained and evaluated.

One important option in the model configuration is the batch size, which determines how many samples are fed to the model at a time during training. A larger batch size can lead to faster training, but may also require more memory. In this case, the batch size is set to 25.

Another important option is the size of the input images. In this case, the images are 25x25 pixels in size and have 3 color channels (RGB). This means that each image consists of 25 rows, 25 columns, and 3 layers of color information.

The model also has a loss function, which is used to measure the error of the model during training. In this case, the loss function is set to sparse categorical crossentropy, which can work with integer targets. This loss function is often used for classification tasks, where the model is trying to predict the class of an input sample.

The number of classes is another important option in the model configuration. This determines the number of different categories that the model can classify samples into. In this case, the number of classes is set to 10.

The number of epochs is another important option. This determines the number of times the model will go through the entire training data set during training. A larger number of epochs can lead to better model performance, but also takes longer to train. In this case, the number of epochs is set to 25, which is relatively low.

The model also has an optimizer, which is used to update the model's parameters based on the loss during training. The Adam optimizer is a popular choice, as it extends traditional gradient descent with local parameter updates and momentum-like optimization.

lastly, the verbosity option determines whether the output of the model training will be displayed on screen. A value of 1 means that the output will be displayed, while a value of 0 means that it will not. In this case, verbosity is set to 1, so the output will be displayed. However, in normal settings, verbosity is often set to 0 to avoid slowing down the training process.

# Data configuration

TRAINING\_SET\_FOLDER = './fruits-360/Training\_smaller'

TEST\_SET\_FOLDER = './fruits-360/Test\_smaller'

# Model configuration

BATCH\_SIZE = 25

IMG\_WIDTH, IMG\_HEIGHT, IMG\_NUM\_CHANNELS = 25, 25, 3

LOSS\_FUNCTION = sparse\_categorical\_crossentropy

NO\_CLASSES = 10

NO\_EPOCHS = 25

OPTIMIZER = Adam()

VERBOSITY = 1

rather than having all the options in one place. This could make it easier to change or update the model configuration in the future, as all the options would be in one place.

def setup\_model\_config(batch\_size, img\_width, img\_height, img\_num\_channels,

loss\_function, no\_classes, no\_epochs, optimizer, verbosity):

"""Set up the model configuration with the given parameters."""

model\_config = {

"batch\_size": batch\_size,

"img\_width": img\_width,

"img\_height": img\_height,

"img\_num\_channels": img\_num\_channels,

"loss\_function": loss\_function,

"no\_classes": no\_classes,

"no\_epochs": no\_epochs,

"optimizer": optimizer,

"verbosity": verbosity

}

return model\_config

**PART IV**

Configuration options are set for data and model. Data paths are set for training and testing data. Model batch size is set to 25, image width and height are set to 25x25 pixels, number of channels is set to 3, loss function is set to sparse categorical crossentropy, number of classes is set to 10, number of epochs is set to 25, and verbosity is set to 1.

The data is loaded and prepared by setting the input shape, creating an ImageDataGenerator object and configuring it to rescale the data. The training data is then fed to the generator using flow\_from\_directory and specifying the training data folder, save directory, batch size, target size, and class mode.

The model architecture is specified using the Sequential model from Keras. It includes an input layer, 3 convolutional layers, 2 max pooling layers, a flatten layer, and a fully connected layer with 10 units for the 10 classes. The model is then compiled using the specified loss function, optimizer, and metrics.

Finally, we will use our trained model to make predictions on new, unseen images of fruit. We will see how well it is able to classify these images and explore ways in which we can improve its performance