Benoît Clemenceau
benclem@bu.edu
U81236938

MET CS 521
Final Project
August 12, 2022

MET CS 521 – Information Structures with Python

Final Project

Image Classification using a Convolutional Neural Network

Benoît Clemenceau
benclem@bu.edu
U81236938

MET CS 521
Final Project
August 12, 2022

Table of Contents

Benoît Clemenceau
benclem@bu.edu
U81236938

MET CS 521
Final Project
August 12, 2022

I. Introduction

My goal for this project was to classify images into their corresponding classes using a Convolutional Neural Network (CNN). My plan was to start with an easy dataset such as Fruits 360, and to classify it using a basic sequential CNN model. Next, I wanted to try with harder ones and I planned on using different techniques to achieve good accuracy. unfortunately, I didn't really have the time to go through with what I wanted to do, but I did get some good results.

II. Fruits Classification

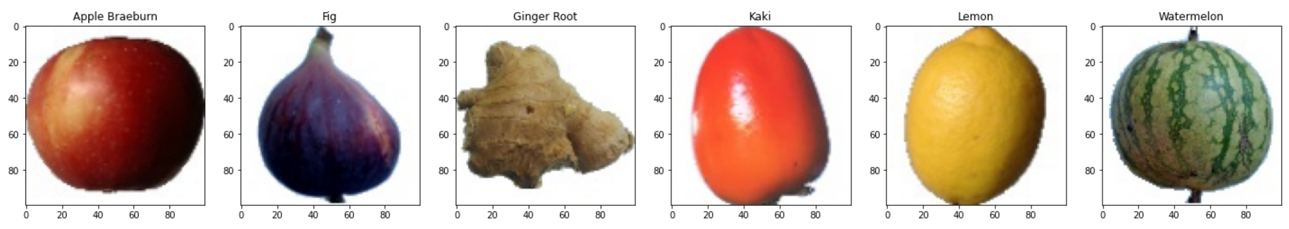
Setup

I started by installing and importing all the packages I needed. I used the "!{sys.executable} -m pip install package" syntax as recommended in this article, because managing packages with Jupyter kernels can get tricky and this is a safer way of doing it. However, this produces a lot of output which I ignored by adding the "%%capture" annotation at the beginning of the cell.

Next, I downloaded the dataset using the opendatasets library, which relies on the Kaggle Official API. It will ask you for your Kaggle API token that you can find in your Kaggle account settings. Kaggle datasets are open source, but downloading them from the API requires authentication.

Data exploration

The images in the dataset look like that:



We have 131 different categories of fruits that are each composed of 600 different pictures on average. These pictures are composed of 100x100 RGB pixels, and are all cropped so that the fruit takes the maximum space possible without losing parts of it. That is why I chose this dataset to start, as it makes the learning easier.

Data preprocessing

When working with images, data preprocessing is very important. The fact that our dataset is so clean is helping a lot, but we still have things to do.

We start by loading all the paths to the pictures and their corresponding category. Then, we actually load the images into PIL format using a Tensorflow Keras utility, and we convert them into 3D NumPy arrays. Our images are now loaded as an array of dimensions: (Number of samples, Image height, Image width, RGB).

Now, to simplify calculation when training the model, we have to scale down the RGB values of the images. We do that by diving every RGB values by 255 (max RGB value). We then have values ranging from 0 to 1.

Finally, we need to make the response vector categorical for the model to be able to compare its predictions with actual values. We do that by using a Keras utility, and we get that kind of result: 'Strawberry' => [0, 0, 0, 1, 0] out of ['Banana', 'Peach', 'Kiwi', 'Strawberry', 'Cherry'].

Building the model

We use a Sequential model which is a plain stack of layers where each layer has exactly one input tensor and one output tensor. We then define the layers we want to add to it one by one.

For the feature extraction part, we added the following layers:

- Conv2D: 2D convolutional layer, extracts low-level features after applying filters to original input to reduce their number.
- Activation: ReLU activation function, decides which neurons should fire. We use ReLU because it works great with CNNs and it's fast.
- MaxPooling2D: 2D MaxPooling layer, reduces dimensionality
- Conv2D: extracts medium-level features

- MaxPooling2D: reduces dimensionality
- Conv2D: extracts high-level features
- MaxPooling2D: reduces dimensionality
- Conv2D: extracts top-level features
- MaxPooling2D: reduces dimensionality

Then for the classification part:

- Dropout: drop some neurons from previous layers to reduce overfitting
- Flatten: converts the pooled 2D array of features into a 1D array
- Dense: Fully connected layer, reduces the number of features
- Activation: decides which neurons should fire
- Dropout: reduces overfitting
- Dense: normalize the output to probabilities for each predicted output classes with a Softmax activation function

We then compile the model with a categorical_crossentropy loss function because we have multiple output categories, and we use Root Mean Squared Propagation as an optimizer function.

Training

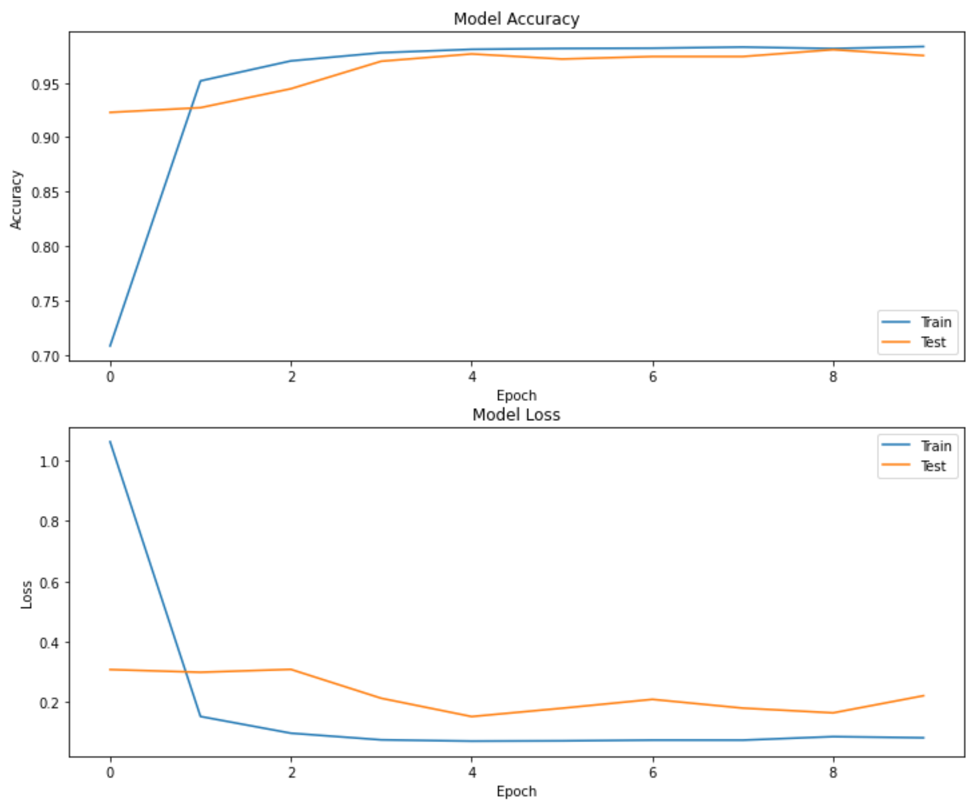
We trained the model on 10 epochs: all the training data went 10 times through the model. For each batch of data going through the model, probabilities of belonging to the classes are calculated, and then compared with the actual class. The result of this comparison tells the model if the weights that were used to calculate these probabilities are valuable or not. As the batches and epochs go by, the weights become more and more adjusted to the problem, and we can measure that trough the validation accuracy statistic.

I trained the model for 10 epochs and that took a little less than an hour on my machine.

Model evaluation

We get a final test accuracy of 97%.
That mean that, when presented with a random image from the dataset, the model is able to tell us which one of the 131 categories of fruits it was trained with it is, while being 97% confident on average.

Here is the evolution of the accuracy and loss for both training and testing data over the epochs:



III. Cats & Dogs

This other classification problem is pretty similar, except for the facts that 1) the response variable is binary (cat or dog) and 2) the images in this dataset are very variable. The size of the pictures varies, the animals are being photographed in all sorts of positions, with different backgrounds and light settings, and with other animals or humans.

Data preprocessing was similar, we just had to normalize the images so that they all have the same dimensions, because the computations in the convolutional layers require constant dimensions. I also had to delete some files in the dataset that should not have been there and were messing with the data loading.

I started by applying a new model identical to the previous one, and got an accuracy of 79% after 10 epochs (≈ 40 mins).

I then tried using transfer learning with a pre-trained ResNet-50 model but didn't get the expected results. I trained it for 5 epochs (≈ 1h) and got a 63% accuracy.