Chapter 5

Deep learning

Python Artificial Intelligence Projects for beginners

Group 22
Almeida Ruas, Fernanda
de Almeida Arruda, Danilo
Freitas, Livia
Machado , Fernando
Mayamba, Thérèse

1. **Introduction**
2. **Project one: Identifying handwritten mathematical symbols** - problematic, approach, step by step
3. **Project two: Revisiting the bird species identifier to use images** - problematic, approach, step by step
4. **Evaluation and feedback**
5. **Conclusion**
6. **Appendix: software and datasets**

1. Introduction

What is Deep Learning?

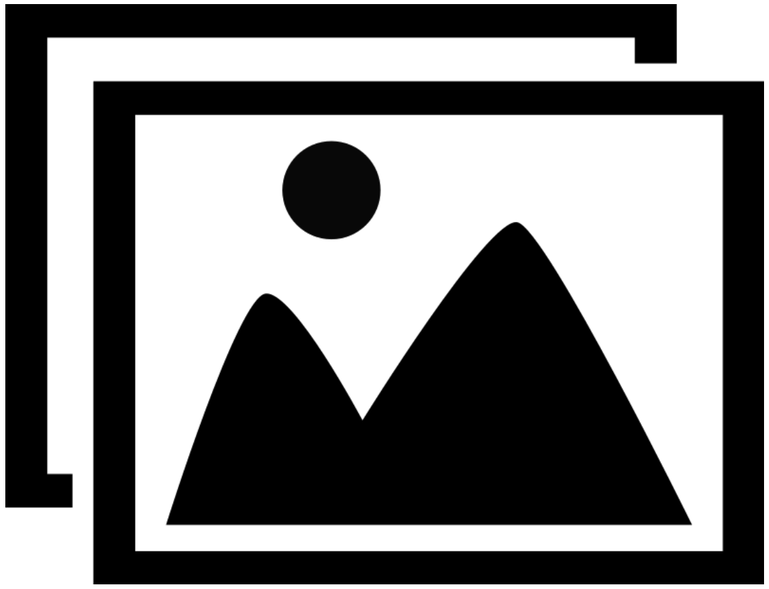
Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans: learn by example. Deep learning is a key technology behind driverless cars, enabling them to recognize a stop sign, or to distinguish a pedestrian from a lamppost. It is the key to voice control in consumer devices like phones, tablets, TVs, and hands-free speakers.
Theorized in the 1980s.

Why is it being more used lately?

1.Deep learning requires large amounts of labelled data.
2. Deep learning requires substantial computing power

It is being used in many sectors:

- *Aerospace and Army : Deep learning is used to identify objects from satellites that locate areas of interest, and identify safe or unsafe zones for troops.*
- *Medical Research: Cancer researchers are using deep learning to automatically detect cancer cells*



s the ground or stays
iverse is vast, and you
s also beautiful. You a
nething bigger than yo
t of something that ma
most of your time. Tak
e a blog post. Make a

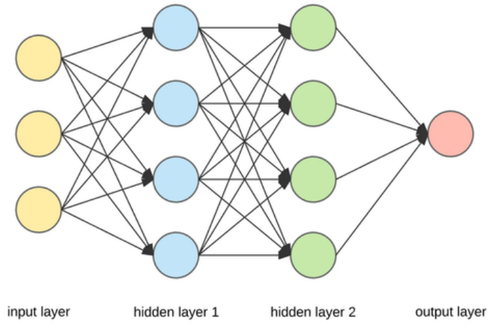
In deep learning, a computer model learns to perform classification tasks directly from images, text, or sound. Deep learning models can achieve state-of-the-art accuracy, sometimes exceeding human-level performance , without the significant effort required for engineering. Models are trained by using a large set of labelled data and neural network architectures that contain many layers.

What is a Neural Network?

A neural network (also called an artificial neural network) is an adaptive system that learns by using interconnected nodes or neurons in a layered structure that resembles a human brain. A neural network can learn from data—so it can be trained to recognize patterns, classify data, and forecast future events. A neural network breaks down the input into layers of abstraction. It can be trained using many examples to recognize patterns in speech or images, for example, just as the human brain does.

We will focus on one of the most powerful algorithms in Deep Learning, the **Convolutional Neural Network (CNN)**, which is a powerful programming model that allows image recognition by automatically assigning a label to each input image corresponding to its class.

How Deep Learning works



- Most deep learning methods use neural network architectures. That is why deep learning models are often referred as deep neural networks.

- The term “deep” usually refers to the number of hidden layers in the neural network. Traditional neural networks only contain 2-3 hidden layers, while deep networks can have as many as 150.

- The neural network takes all of the training data in the input layer. Then it passes the data through the hidden layers, transforming the values based on the weights at each node. Finally it returns a value in the output layer.

Source: https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6

Convolutional Neural Networks

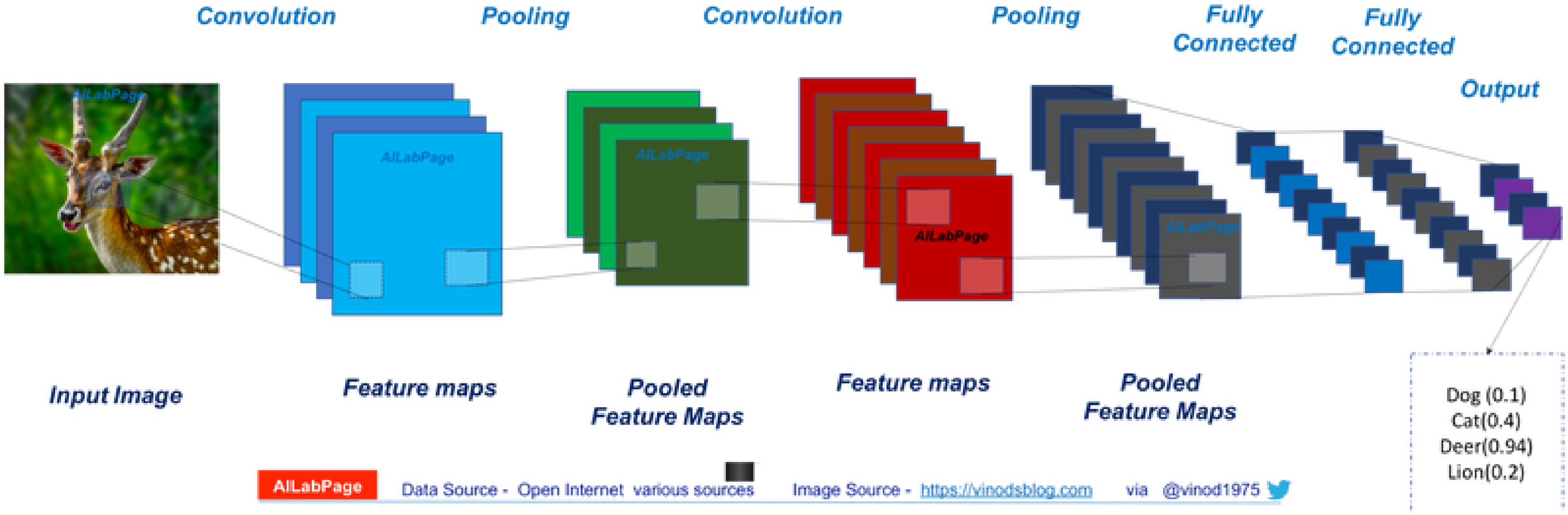
- Convolutional neural networks work by applying filters, to an image in order to understand essential features. Convolutional neural networks (CNNs) are usually used in situations where breaking down the data into parts can make it easier for the algorithm to process it.

- They are made of layers of artificial neurons called nodes. These nodes are functions that calculate the weighted sum of the inputs and return an activation map. This is the convolution part of the neural network.

- Each node in a layer is defined by its weight values. When you give a layer some data, like an image, it takes the pixel values and picks out some of the visual features.

- They usually are built using :
  - **Convolutional layers** that act as filters that check the entire image for information.
  - **Pooling layers** that further act to compress the information. Pooling layers aggregate the convolutional layers' results by doing operations such as selecting the max value in an area (max pooling) or averaging values (average pooling). It only returns the most relevant features from the layer in the activation map.

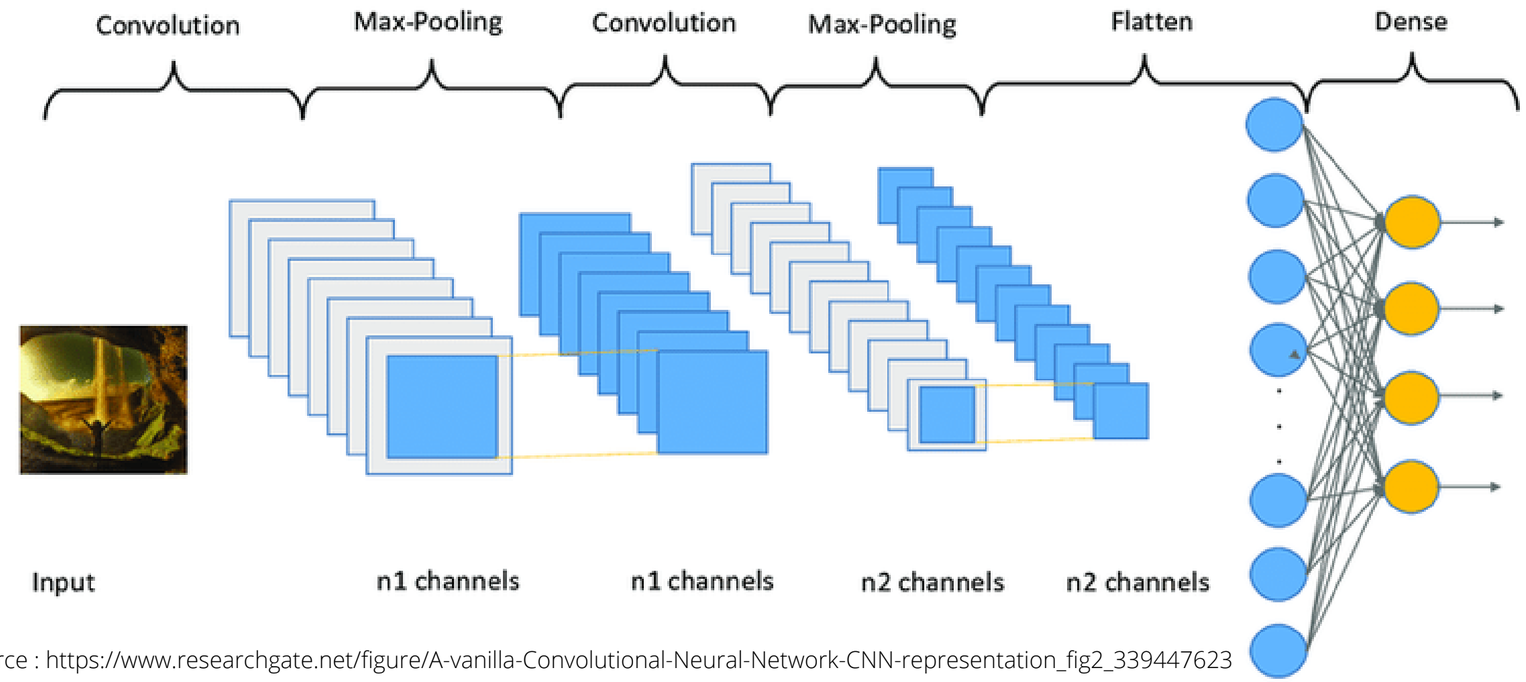
Convolution Neural Network



AILabPage Data Source - Open Internet various sources Image Source - https://vinodsblog.com via @vinod1975

What a convolutional neural network (CNN) does differently

A convolutional neural network is a specific kind of neural network with multiple layers. It processes data that has a grid-like arrangement then extracts important features. One huge advantage of using CNNs is that you don't need to do a lot of pre-processing on images.



Source : https://www.researchgate.net/figure/A-vanilla-Convolutional-Neural-Network-CNN-representation_fig2_339447623

- CNNs can learn what characteristics in the filters are the most important. That saves a lot of time and trial and error work since we don't need as many parameters.

- It doesn't seem like a huge savings until you are working with high resolution images that have thousands of pixels. The convolutional neural network algorithm's main purpose is to get data into forms that are easier to process without losing the features that are important for figuring out what the data represents. This also makes them great candidates for handling huge datasets.

- A big difference between a CNN and a regular neural network is that CNNs use convolutions to handle the math behind the scenes. A convolution is used instead of matrix multiplication in at least one layer of the CNN. Convolutions take to two functions and return a function.

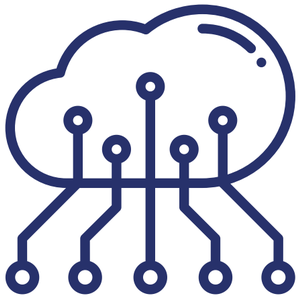
Inputs and Outputs – How it works

- In CNNs it requires to models to train and test. Each input image pass through a series of convolution layers with filters (**Kernals**), Pooling, **fully connected layers** (FC) and apply softmax function (Generalisation of the logistic function that "**squashes**" a K-dimensional vector of arbitrary real values to real values Kd vector) to classify an object with probabilistic values between 0 and 1. This is the reason every image in CCN gets represented as a matrix of pixel values.

- The Convolutional Neural Network classifies an input image into categories **e.g dog, cat, deer, lion or bird.**

- **The Convolution + Pooling** layers act as feature extractors from the input image while fully connected layer acts as a classifier.

- In above image figure, on receiving a dear image as input, the network correctly assigns the highest probability for it **(0.94)** among all four categories. The sum of all probabilities in the output layer should be one though. There are four main operations in the ConvNet shown in image above:



Convolutional Neural Network-CNN architecture

CNNs are a sub-category of neural networks and are currently one of the most powerful image classification models.

At first sight, their mode of operation is simple: the user provides an input image in the form of a matrix of pixels. This has 3 dimensions:

- Two dimensions for a greyscale image.
- A third dimension, of depth 3, to represent the fundamental colours (Red, Green, Blue).

Contrary to a classic MLP (Multi Layers Perceptron) model which only contains a classification part, the Convolutional Neural Network architecture has a convolutional part upstream and consequently comprises two distinct parts:

- **A convolutional part**: Its final objective is to **extract specific characteristics from each image** by **compressing** them so as **to reduce their initial size**. In short, the input image is passed through a **succession of filters**, creating new images called **convolution maps**. Finally, the resulting **convolution maps are concatenated** into a feature vector called the CNN code.
- **A classification part**: **The CNN code obtained at the output** of the convolutional part is **provided as input in a second part,** consisting of fully connected layers called Multi Layers Perceptron (MLP). The role of this part is to **combine the features of the CNN code in order to classify the image.**

Project #1

Identifying handwritten mathematical symbols

Approach: CNNs

Problematic

For this first project, we will need to build a CNN (Convolutional Neural Network) system which will be able to read and classify handwritten mathematical symbols.

In order to do so, we will use Jupyter notebook (Python programing language) and a series of popular libraries for machine learning using Python.

The above resources will allow us to draw an approach towards our main goal: **having our prediction model to identify the confidence of our entered inputs (the mathematical symbols).**

Approach

The approach we used to build the requested deep learning model can basically be divided into four phases:

First phase: data collection and import

- First of all, we will be using **Kaggle** as it allows us to work online with Jupyter Notebook for computers with limited space capacity;
- We will collect an exisiting dataset of handwritten mathematical symbols called **HASYv2**. This file contains 168,000 images from 369 classes where each represents a different symbol. Importing this dataset into Jupyter notebook is the first step as it will allow us to train our machine learning model;
- We will also need to import three python libraries:

1 - CSV (Comma-Separated Values): Its main function is to store data in a tabular format, which will further allow us to easily share data between two different applications;
2 - PIL: This library will be an essential ellement to manipulate imagens in Python;
3 - KERAS: This is a very helpfull library which will allow us to **standardize the images we imported**. When working with machine learning, it is essential that images have the same size and shape. This will be a key step when building and training our deep learning model

Approach

Second phase: Model selection

The book's chapter requests to apply CNN to build our deep learning model. CNN will help us to classify our mathematical symbols based on features we extracted in the previous phase.

But, why CNN?

A CNN is ideal for our project because it uses multiple convolutional layers. This is especially useful for cases like ours where we are processing many different symbols and images which may share common features.
In short, CNNs are ideal to build our model due to the **precision to capture spatial information** and **process invariances as rotation and scaling.**

Below an example of how CNN could capture the different features (instead of other models):

Stewart, M., PhD. (2021, 8 décembre)

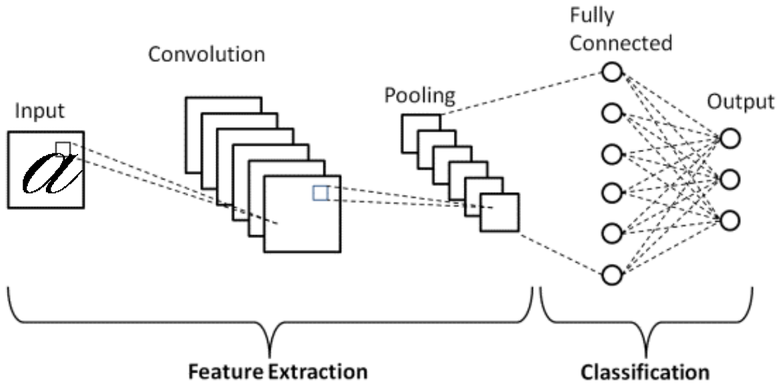
Approach

Third phase: features extraction (model training)

This is a very important step to successfully be able to solve our problem. Despite the fact that CNNs are designed to learn and extract information from the images we will input, it is still important to preprocess those images.
This will bring more **assertiveness** to our results and contribute to the **performance of our built network**.

We can summarize the features extraction steps we will follow as:

- **Image Preprocessing:** There are several techiques, but for our model **only resizing** will be required;
- **Relevant features extracting:** This consists essentially of using **convolutional layers** (learn and extract features)**, pooling layers** (used to downsample / reduce network compexity) and finally **fully connected layers** will be used to classify our processed features before our **final input.**



Approach

Fourth phase: classification & evaluation

Once we finish features extraction from the previous phase we proceed to the **classification task**, which will be in charge of assigning lables to the input data based on the features our model learned in the previous phase.
When it comes to building mathematical symbols detection models (as in our case), the classification task will allow us to **predict the accurate symbol class** (such as "plus", "minus", "integral" etc.).
By doing so, our CNN model will map the features we extracted in the previous phase to correspond to the correct symbol label.

Finally, the **evaluation phase** will allow us to teste **the accuracy and precision** of the model we built**:**

- **Accuracy:** the percentage of correctly classified images **(confidence ratio)**
- **Precision:** the share of true positive grades amongst all positive grades.

Solution

Create the system - a step by step demonstration

1 – We begin by adding the code that allows us to import any image.

[1]:
```
# La classe Image permet de charger et d'afficher des images dans un notebook Jupyter
from IPython.display import Image
```

2 – We imported an image from the library, this code allows us to show images. In this example the letter A. PS: we needed to adjust the code, since we are using Kaggle.

```
# Permet d'afficher l'image
Image("/kaggle/input/hasyv2/hasy-data/v2-00010.png")
```

After running the code, this is the result. The image is 30x30 pixels with three different channels (RGB).

3 – Next, we imported three necessary libraries to work with CSV files. CSV allow us to load the necessary dataset.

```
import csv
from PIL import Image as pil_image
import keras.preprocessing.image
```

Where:

- The CSV (Comma Separated Values) is a popular file format used to store tabular data.
- PIL (Python Imaging Library) library provides tools for image manipulation and processing.
- The third line "keras.preprocessing.image" allows us to convert the images in "numpy arrays" - a type of data structure.

4 – Next step is commonly done before training a deep learning model on image data. In the code below, we go through every file name and record which class it belongs to.

Unlike the book, we decided to add this part in the code "if classes[img_class] < 50" to keep only the 50 images of each character, thereby reducing the time for running the code and the RAM used in the process.

```
imgs = []
classes = {}
with open('/kaggle/input/hasyv2/hasy-data-labels.csv') as csvfile:
    csvreader = csv.reader(csvfile)
    i = 0
    for row in csvreader:
        if i > 0:
            img_class = row[2]
            if img_class not in classes:
                classes[img_class] = 0
            if classes[img_class] < 50:
                img = keras.preprocessing.image.image_utils.img_to_array(pil_image.open("/kaggle/input/hasyv2/" + row[0]))
                img /= 255.0
                imgs.append((row[0], img_class, img))
                classes[img_class] += 1
        i += 1

classes = list(classes.keys())
```

5 – Here, we set a counter to make sure the first row is not considered, since it is the header of the CSV file. Then, we open the image, represented in the first column of each row, and all are converted into an array. The results have dimensions of 30 widths, 30 heights, and 3 (which represent the RGB channels).

The problem is that these channels will have typical pixel results between 0 and 255, which are not good for neural networks. The solution is to divide each pixel value by 255.

```
imgs[0]
```

```
[5]: ('hasy-data/v2-00000.png',
      'A',
      array([[[1., 1., 1.],
              [1., 1., 1.],
              [1., 1., 1.],
              ...,
              [1., 1., 1.],
              [1., 1., 1.],
              [1., 1., 1.]],

             [[1., 1., 1.],
              [1., 1., 1.],
              [1., 1., 1.],
              ...,
              [1., 1., 1.],
              [1., 1., 1.],
              [1., 1., 1.]],

             [[1., 1., 1.],
              [1., 1., 1.],
              [1., 1., 1.],
              ...,
              [1., 1., 1.],
              [1., 1., 1.],
              [1., 1., 1.]],

             ...,

             [[1., 1., 1.],
              [1., 1., 1.],
              [1., 1., 1.],
              ...,
              [1., 1., 1.],
              [1., 1., 1.],
              [1., 1., 1.]],

             [[1., 1., 1.],
              [1., 1., 1.],
              [1., 1., 1.],
              ...,
              [1., 1., 1.],
              [1., 1., 1.],
              [1., 1., 1.]],

             [[1., 1., 1.],
              [1., 1., 1.],
              [1., 1., 1.],
              ...,
              [1., 1., 1.],
              [1., 1., 1.],
              [1., 1., 1.]]], dtype=float32))
```

After all, we collect the filename, the class name, and the image matrix to all image list that can be used as input to a deep learning model. Each 1. represents the white color

6 – The code below outputs the number of images in the HASYv2 dataset.

Unlike the book, we had a result of 18450 images loaded and passed through the preprocessing steps because we limited the images to 50. In the book, the number was 168233.

[6]:
```
len(imgs)
```

[6]: 18450

7 – Next, the code below shuffles the imgs list, then splits it into a training set of 80% and a 20% test set.

This step ensures that all the images are randomly distributed between the two subsets, which is important for avoiding bias in the model training and evaluation.

[7]:
```
import random

random.shuffle(imgs)
split_idx = int(0.8*len(imgs))
train = imgs[:split_idx]
test = imgs[split_idx:]
```

8 – Since we have tuples with three different values, we collect everything (labels, images) and put them all together in an array, taking each third element of each row. Next, we do the same with train datasets.

The outputs are collected by taking the second value. However, to be able to use them in a neural network, we need to convert them into a hot-encoding.

[8]:
```
import numpy as np

train_input = np.asarray(list(map(lambda row: row[2], train)))
test_input = np.asarray(list(map(lambda row: row[2], test)))

train_output = np.asarray(list(map(lambda row: row[1], train)))
test_output = np.asarray(list(map(lambda row: row[1], test)))
```

9 – To do so, we use a sklearn processing label encoder and an one hote encoder. The goal of using them is:

- LabelEncoder is used to transform non-numerical labels into numerical labels.
- OneHotEncoder is used to transform categorical labels to one-hot encoded binary vectors.

[9]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
```

10 – Next step is to make a label Encoder object and fit and transform in classes.

The code used aims to convert class names into a one-hot encoding format (conversion of categorical variables into numerical format to be used in machine learning models).

- First, the LabelEncoder class is used to encode the class names into integers.
- The integer encoded class labels are then converted to one-hot encoding using the OneHotEncoder class.
- The integer-encoded class labels of the "train_output" training set are stored in the "train_output_int" variable, while the integer-encoded class labels of the test_output" are stored in the variable "test_output_int".
- These integer-encoded class labels are then converted to one-hot encoding using the OneHotEncoder class. The one-hot encoded class labels of the training set "train_output" are stored in the "train_output" variable, while the one-hot encoded class labels of the test set "test_output" are stored in the "test_output" variable.
- The number of classes is determined using the classes_ method of the LabelEncoder class and is stored in the "num_classes" variable. This value is used later on in the definition of the machine learning model.

```
# first, convert class names into integers
label_encoder = LabelEncoder()
integer_encoded = label_encoder.fit_transform(classes)

# then convert integers into one-hot encoding
onehot_encoder = OneHotEncoder(sparse=False)
integer_encoded = integer_encoded.reshape(len(integer_encoded), 1)
onehot_encoder.fit(integer_encoded)

# convert train and test output to one-hot
train_output_int = label_encoder.transform(train_output)
train_output = onehot_encoder.transform(train_output_int.reshape(len(train_output_int), 1))
test_output_int = label_encoder.transform(test_output)
test_output = onehot_encoder.transform(test_output_int.reshape(len(test_output_int), 1))

num_classes = len(label_encoder.classes_)
print("Number of classes: %d" % num_classes)
```

```
Number of classes: 369
```

11 – Now, the next step is to import the neural network layers from the Keras library since we will need to build our deep learning model.

The imported layers are:
Sequential: a linear stack of neural network layers, which will allow us to simply add layers on top of each other.
Dense: a fully connected neural network layer, where each input is connected to each output.
Dropout: a regularization technique that will allow us to randomly disable some neurons some neurons during training to avoid overfitting.
Flatten: a layer that transforms an image into a 1D array.
Conv2D: a convolution layer that applies a 2D convolution filter to the input image.
MaxPooling2D: a pooling layer that extracts the most important features of the image by reducing its dimensions.

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
```

12 – **This code creates a convolutional neural network (CNN) model in Keras.**

Here is a step by step short description of the code:

1. We created a sequential object using the Sequential() function from Keras.
2. The first layer in our model is a Conv2D layer with 32 filters, a kernel size of (3,3), and a ReLU activation function.
3. A MaxPooling2D layer was added to the model.
4. Another Conv2D layer was added to the model with the same figures we entered for the layer 2.
5. We added another MaxPooling2D layer (pool size of (2,2)).
6. A Flatten layer was added to convert the output of the previous layers into a 1-dimensional array.
7. A Dense layer containing 1024 neurons and a hyperbolic tangent (tanh) activation function was also added.
8. The we proceeded to add a dropout layer with a rate of 0.5 (to prevent overfitting)
9. Finally, a Dense layer with num_classes neurons and a softmax activation function was added to the model. This is the output layer of the model.
10. The model was then compiled with the loss function categorical_crossentropy, the optimizer Adam optimizer and the accuracy metric.
11. The model.summary() method displays a summary of the model architecture, including the number of trainable parameters.

[12]:
```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3,3), activation="relu", input_shape=np.shape(train_input[0])))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(32, (3, 3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(1024, activation="tanh"))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation="softmax"))

model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 max_pooling2d (MaxPooling2D  (None, 15, 15, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 13, 13, 32)        9248

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 32)         0
 2D)

 flatten (Flatten)           (None, 1152)              0

 dense (Dense)               (None, 1024)              1180672

 dropout (Dropout)           (None, 1024)              0

 dense_1 (Dense)             (None, 369)               378225

=================================================================
Total params: 1,569,041
Trainable params: 1,569,041
Non-trainable params: 0
_________________________________________________________________
None
```

13 – This code imports the Keras callbacks module and creates a TensorBoard callback object that will be used to log data during training. The logs will be stored in the "./logs/mnist-style" directory.

[13]:
```
import keras.callbacks

tensorboard = keras.callbacks.TensorBoard(log_dir="./logs/mnist-style")
```

14 – The goal of the below code is to train a Keras deep learning model using the entered functions.
This code will adjust the model weights so that the loss function is minimized and the accuracy is maximized on the training data.

[14]:
```
model.fit(train_input, train_output, batch_size=32, epochs=10, verbose=2, validation_split=0.2, callbacks=[tensorboard])
```

```
Epoch 1/10
369/369 - 11s - loss: 3.8463 - accuracy: 0.2330 - val_loss: 2.3310 - val_accuracy: 0.4394 - 11s/epoch - 30ms/step
Epoch 2/10
369/369 - 10s - loss: 1.9140 - accuracy: 0.5145 - val_loss: 1.8789 - val_accuracy: 0.5352 - 10s/epoch - 26ms/step
Epoch 3/10
369/369 - 9s - loss: 1.4226 - accuracy: 0.6144 - val_loss: 1.7572 - val_accuracy: 0.5400 - 9s/epoch - 26ms/step
Epoch 4/10
369/369 - 10s - loss: 1.1004 - accuracy: 0.6824 - val_loss: 1.7837 - val_accuracy: 0.5596 - 10s/epoch - 26ms/step
Epoch 5/10
369/369 - 10s - loss: 0.8765 - accuracy: 0.7376 - val_loss: 1.7951 - val_accuracy: 0.5606 - 10s/epoch - 26ms/step
Epoch 6/10
369/369 - 10s - loss: 0.6911 - accuracy: 0.7884 - val_loss: 1.8796 - val_accuracy: 0.5657 - 10s/epoch - 26ms/step
Epoch 7/10
369/369 - 10s - loss: 0.5869 - accuracy: 0.8207 - val_loss: 1.9946 - val_accuracy: 0.5623 - 10s/epoch - 28ms/step
Epoch 8/10
369/369 - 10s - loss: 0.4904 - accuracy: 0.8475 - val_loss: 2.0154 - val_accuracy: 0.5684 - 10s/epoch - 27ms/step
Epoch 9/10
369/369 - 10s - loss: 0.4212 - accuracy: 0.8688 - val_loss: 2.1423 - val_accuracy: 0.5589 - 10s/epoch - 28ms/step
Epoch 10/10
369/369 - 9s - loss: 0.3617 - accuracy: 0.8866 - val_loss: 2.2388 - val_accuracy: 0.5566 - 9s/epoch - 25ms/step
```

[14…]:
```
<keras.callbacks.History at 0x7fc1d460a750>
```

15 – When entering this code, we will be performing a **hyperparameter** search over a range of **convolutional layer counts, dense layer sizes, and dropout rates to train multiple Keras deep learning models.**

For each combination of hyperparameters, a model is trained and its validation loss and accuracy are logged to TensorBoard.

After each model is trained, the loss, accuracy, and training time are printed to the console.

[15]:
```
import time

results = []

for conv2d_count in [1,2]:
    for dense_size in [128,256,512]:
        for dropout in [0.0, 0.25, 0.50, 0.75]:
            model = Sequential()

            for i in range(conv2d_count):
                if i == 0:
                    model.add(Conv2D(32, kernel_size=(3,3), activation="relu", input_shape=np.shape(train_input[0])))
                else:
                    model.add(Conv2D(32, kernel_size=(3,3), activation="relu"))
                model.add(MaxPooling2D(pool_size=(2,2)))
            model.add(Flatten())
            model.add(Dense(dense_size, activation="tanh"))

            if dropout > 0.0:
                model.add(Dropout(dropout))
            model.add(Dense(num_classes, activation="softmax"))

            model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

            log_dir = "./logs/conv2d_%d-dense_%d-dropout_%.2f" % (conv2d_count, dense_size, dropout)
            tensorboard = keras.callbacks.TensorBoard(log_dir=log_dir)

            start = time.time()
            model.fit(train_input, train_output, batch_size=32, epochs=10, verbose=0, validation_split=0.2, callbacks=[tensorboard])
            score = model.evaluate(test_input, test_output, verbose=2)
            end = time.time()

            print("Conv2D count: %d, Dense size: %d, Dropout: %.2f - Loss: %.2f, Accuracy: %.2f, Time: %d sec" % (conv2d_count, dense_size, dropout, score[0], score[1], end - start))
```

```
116/116 - 1s - loss: 1.6652 - accuracy: 0.5791 - 727ms/epoch - 6ms/step
Conv2D count: 1, Dense size: 128, Dropout: 0.00 - Loss: 1.67, Accuracy: 0.58, Time: 60 sec
116/116 - 1s - loss: 1.6111 - accuracy: 0.5780 - 696ms/epoch - 6ms/step
Conv2D count: 1, Dense size: 128, Dropout: 0.25 - Loss: 1.61, Accuracy: 0.58, Time: 60 sec
116/116 - 1s - loss: 1.5391 - accuracy: 0.5913 - 665ms/epoch - 6ms/step
Conv2D count: 1, Dense size: 128, Dropout: 0.50 - Loss: 1.54, Accuracy: 0.59, Time: 61 sec
116/116 - 1s - loss: 1.6130 - accuracy: 0.5932 - 674ms/epoch - 6ms/step
Conv2D count: 1, Dense size: 128, Dropout: 0.75 - Loss: 1.61, Accuracy: 0.59, Time: 61 sec
116/116 - 1s - loss: 1.6625 - accuracy: 0.5837 - 823ms/epoch - 7ms/step
Conv2D count: 1, Dense size: 256, Dropout: 0.00 - Loss: 1.66, Accuracy: 0.58, Time: 83 sec
116/116 - 1s - loss: 1.6183 - accuracy: 0.5965 - 776ms/epoch - 7ms/step
Conv2D count: 1, Dense size: 256, Dropout: 0.25 - Loss: 1.62, Accuracy: 0.60, Time: 77 sec
116/116 - 1s - loss: 1.5703 - accuracy: 0.5935 - 791ms/epoch - 7ms/step
```

16 – To have the best-trained model, we are going to retrain all the data.

For doing so, we will use this code to define, train, and save a CNN model for classifying images of math symbols, which can later be used for making predictions on new images.

Here we have two convolution layers, our dense is 128 dropout 0.5. All the data trained and tested before are all together.

[16]:

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3,3), activation="relu", input_shape=np.shape(train_input[0])))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(32, (3,3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(128, activation="tanh"))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation="softmax"))
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
print(model.summary())

# join train and test data so we train the network on all data we have available to us
model.fit(np.concatenate((train_input, test_input)), np.concatenate((train_output, test_output)), batch_size=32, epochs=10, verbose=2)

# save the trained model
model.save("mathsymbols.model")

# save label encoder (to reverse one-hot encoding)
np.save("classes.npy", label_encoder.classes_)
```

```
Model: "sequential_25"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_38 (Conv2D)          (None, 30, 30, 32)        896

 max_pooling2d_38 (MaxPoolin  (None, 15, 15, 32)       0
 g2D)

 conv2d_39 (Conv2D)          (None, 13, 13, 32)        9248

 max_pooling2d_39 (MaxPoolin  (None, 6, 6, 32)         0
 g2D)

 flatten_25 (Flatten)        (None, 1152)              0

 dense_50 (Dense)            (None, 128)               147584

 dropout_19 (Dropout)        (None, 128)               0

 dense_51 (Dense)            (None, 369)               47601

=================================================================
Total params: 205,329
Trainable params: 205,329
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
577/577 - 10s - loss: 4.6652 - accuracy: 0.1225 - 10s/epoch - 18ms/step
Epoch 2/10
577/577 - 9s - loss: 2.8204 - accuracy: 0.3653 - 9s/epoch - 16ms/step
Epoch 3/10
577/577 - 9s - loss: 2.1629 - accuracy: 0.4811 - 9s/epoch - 16ms/step
Epoch 4/10
577/577 - 9s - loss: 1.8289 - accuracy: 0.5372 - 9s/epoch - 16ms/step
Epoch 5/10
577/577 - 9s - loss: 1.6273 - accuracy: 0.5764 - 9s/epoch - 16ms/step
Epoch 6/10
577/577 - 9s - loss: 1.4811 - accuracy: 0.6031 - 9s/epoch - 16ms/step
Epoch 7/10
577/577 - 9s - loss: 1.3785 - accuracy: 0.6199 - 9s/epoch - 16ms/step
Epoch 8/10
577/577 - 9s - loss: 1.3004 - accuracy: 0.6343 - 9s/epoch - 16ms/step
Epoch 9/10
577/577 - 9s - loss: 1.2270 - accuracy: 0.6506 - 9s/epoch - 16ms/step
Epoch 10/10
577/577 - 9s - loss: 1.1647 - accuracy: 0.6622 - 9s/epoch - 16ms/step
```

17 – Next, we are using this code to load a pre-trained Keras model from a file named "mathsymbols.model".

1. It prints a summary of the model architecture using the summary() function.
2. It restores the class name to integer encoder from a previously saved numpy file named "classes.npy".
3. It defines a function named "predict" that takes an image file path as input.
4. The function uses the Pillow library to load the image file as a numpy array, which is then normalized by dividing by 255.0.
5. The normalized image is then passed through the pre-trained model to obtain a prediction.
6. The prediction is a one-hot encoded vector of probabilities, and the function uses argmax to find the index of the highest-scoring output neuron.
7. The index is then reverse transformed using the label_encoder2 object to get the predicted class name, which is printed along with the prediction confidence.

The end goal of this code is to use a pre-trained model to make predictions on new images of math symbols, given as input to the predict function.

[20]:

```
# Load the pre-trained model and predict the math symbol for an arbitrary image; the code below could be placed in a separate file

import keras.models

model2 = keras.models.load_model("mathsymbols.model")
print(model2.summary())

# restore the class name to integer encoder
label_encoder2 = LabelEncoder()
label_encoder2.classes_ = np.load("classes.npy")

def predict(img_path):
    newimg = keras.preprocessing.image.image_utils.img_to_array(pil_image.open(img_path))
    newimg /= 255.0

    # do the prediction
    prediction = model2.predict(newimg.reshape(1, 32, 32, 3))

    # figure out whitch output neuron had the highest score, and reverse the one-hot encoding
    inverted = label_encoder2.inverse_transform([np.argmax(prediction)]) # argmax finds highest-scoring output
    print("Prediction: %s, confidence: %.2f" % (inverted[0], np.max(prediction)))
```

```
Model: "sequential_25"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_38 (Conv2D)          (None, 30, 30, 32)        896

 max_pooling2d_38 (MaxPoolin  (None, 15, 15, 32)       0
 g2D)

 conv2d_39 (Conv2D)          (None, 13, 13, 32)        9248

 max_pooling2d_39 (MaxPoolin  (None, 6, 6, 32)         0
 g2D)

 flatten_25 (Flatten)        (None, 1152)              0

 dense_50 (Dense)            (None, 128)               147584

 dropout_19 (Dropout)        (None, 128)               0

 dense_51 (Dense)            (None, 369)               47601

=================================================================
Total params: 205,329
Trainable params: 205,329
Non-trainable params: 0
_________________________________________________________________
None
```

18 – Finally, we use the code below to make a prediction on a given image of a math symbol after our pre-trained deep learning model. we obtained the following results:

- The prediction for A is 0.26 or 26%
- The prediction for pi is 0.36 or 36%
- The prediction for alpha is 0.48 or 48%

Our results differ from the book because we limited the number of images used to 50.

```
In [18]: predict("/kaggle/input/hasyv2/hasy-data/v2-00010.png")

         1/1 [==============================] - 0s 115ms/step
         Prediction: A, confidence: 0.26

In [19]: predict("/kaggle/input/hasyv2/hasy-data/v2-00500.png")

         1/1 [==============================] - 0s 22ms/step
         Prediction: \pi, confidence: 0.36

In [20]: predict("/kaggle/input/hasyv2/hasy-data/v2-00700.png")

         1/1 [==============================] - 0s 22ms/step
         Prediction: \alpha, confidence: 0.48
```

Project #2

Revisiting the bird species identifier to use images

Problematic

Using the CNN network to predict bird images.

Approach

In this section, the book used the bird species identifier (second chapter) and will demonstrate how it is possible to update the database and use the neural network in deep learning

In this second moment, the book proposes to use the real images without any pre-processing **without labels, only pictures**. It suggests building a custom convolutional neural network just like it was done for the mathematical symbols classifier.

Using the CNN network through images of birds, with the final objective that it can distinguish the types of birds existing in the database and in the end be able to recognize and distinguish the birds.

Activities are divided into 03 large sessions:

- CNN
- Train
- Predict

CNN

We started working on a virgin neural network, with an average of 720 images, so we used the data generator, tool to multiply these images and make the CNN network have a greater number of images so that it could learn

1- **Importing the Bookstore - Keras**

Problem of libraries (keras), code cannot be executed neither in Jupyter online, or in Jupyter locally installed with Anaconda:

[1]:
```
import numpy as np
from keras.models import Sequential, load_model
from keras.layers import Dropout, Flatten, Conv2D, MaxPooling2D, Dense, Activation
from keras.utils import np_utils
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import TensorBoard
import itertools
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[1], line 2
      1 import numpy as np
----> 2 from keras.models import Sequential, load_model
      3 from keras.layers import Dropout, Flatten, Conv2D, MaxPooling2D, Dense, Activation
      4 from keras.utils import np_utils

ModuleNotFoundError: No module named 'keras'
```

Forum with solutions:
https://stackoverflow.com/questions/56641165/modulenotfounderror-no-module-named-keras-for-jupyter-notebook

Solved, Running in Anaconda CLI:

installation command - pip install keras / pip install tensorflow / conda install keras

Entrée [ ]:
```
import numpy as np
from keras.models import Sequential, load_model
from keras.layers import Dropout, Flatten, Conv2D, MaxPooling2D, Dense, Activation
from keras.utils import np_utils
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import TensorBoard
import itertools
```

We standardize the images: in 3 channels, green red and blue

Entrée [ ]:
```
# all images will be converted to this size
ROWS = 256
COLS = 256
CHANNELS = 3
```

Train image generator

Entrée [3]:
```
train_image_generator = ImageDataGenerator(horizontal_flip=True, rescale=1./255, rotation_range=45)
test_image_generator = ImageDataGenerator(horizontal_flip=False, rescale=1./255, rotation_range=0)

train_generator = train_image_generator.flow_from_directory('trains', target_size=(ROWS, COLS), class_mode='categorical')
test_generator = test_image_generator.flow_from_directory('tests', target_size=(ROWS, COLS), class_mode='categorical')
```

```
Found 11788 images belonging to 200 classes.
Found 11788 images belonging to 200 classes.
```

CNN - Configuration - The same as the book found

```
=================================================================
Total params: 5,210,776
Trainable params: 5,210,776
Non-trainable params: 0
_________________________________________________________________
```

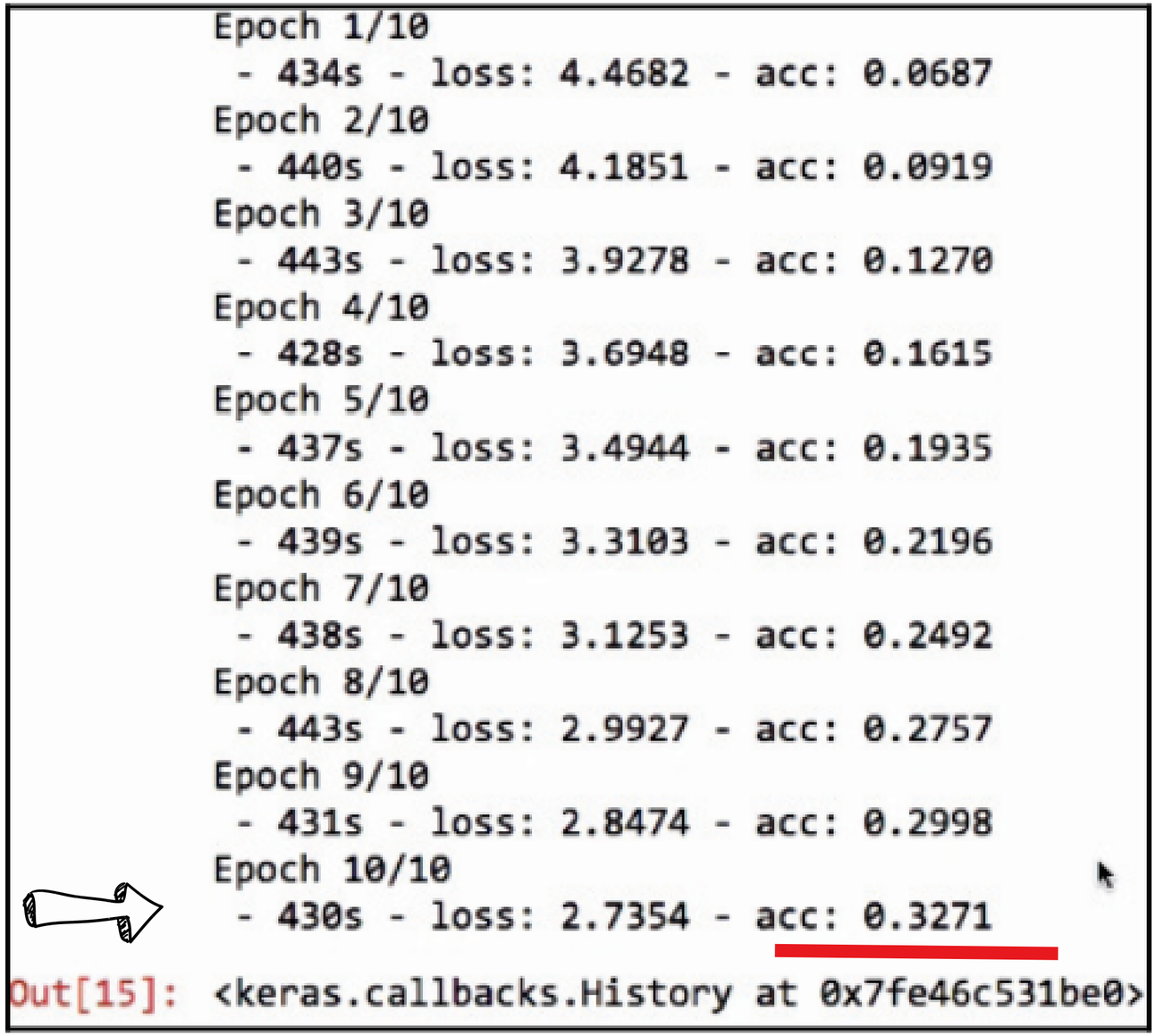
The training step, making an input in the neural network did not work, possibly due to a problem with the number of parameters used - steps-per-epoch 512.

```
: tensorboard = TensorBoard(log_dir='./logs/custom')

model.fit_generator(train_generator, steps_per_epoch=512, epochs=10, callbacks=[tensorboard], verbose=2)
```

```
C:\Users\FernandaAlmeidaRuas\AppData\Local\Temp\ipykernel_108400\447269938.py:3: UserWarning: `Model.fit_generator`
ed and will be removed in a future version. Please use `Model.fit`, which supports generators.
  model.fit_generator(train_generator, steps_per_epoch=512, epochs=10, callbacks=[tensorboard], verbose=2)

Epoch 1/10
WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your dataset or generator can g
least `steps_per_epoch * epochs` batches (in this case, 5120 batches). You may need to use the repeat() function whe
your dataset.
512/512 - 2583s - loss: 5.2910 - accuracy: 0.0061 - 2583s/epoch - 5s/step
```

```
: <keras.callbacks.History at 0x1c55004aa00>
```

we should have arrived to the final value of accuracy of 32%, as a result of the book, shown in the image.



Train

- We will use the CNN trained already for ImageNet for competition, with millions of images and 1000 types of birds… but instead of using all the network, **we will remove the last part, FC (fully-connected) and put ours.**

- **We will benefit of the training at the 1st part and last part we will adapt for our bird case (200 types).**

We use the Inception v3 trained base and only include the layer **- Fully-Connected.**

By doing this it adapts for the 200 types of birds that match the data we are using.

If you left FC of inceptionv3 We would have **1000 types of birds.**

```
# create the base pre-trained model
base_model = InceptionV3(weights='imagenet', include_top=False)

# add a global spatial average pooling layer
x = base_model.output
x = GlobalAveragePooling2D()(x)
# add a fully-connected layer
x = Dense(1024, activation='relu')(x)
out_layer = Dense(200, activation='softmax')(x)

# this is the model we will train
model = Model(inputs=base_model.input, outputs=out_layer)
```

Final configuration of our neural network

Entrée [5]:
```
# first: train only the top layers (which were randomly initialized)
# i.e. freeze all convolutional InceptionV3 layers
for layer in base_model.layers:
    layer.trainable = False

model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])

model.summary()
```

```
 mixed10 (Concatenate)          (None, None, None,    0           ['activation_85[0][0]',
                                2048)                             'mixed9_1[0][0]',
                                                                  'concatenate_1[0][0]',
                                                                  'activation_93[0][0]']

 global_average_pooling2d (Glob  (None, 2048)        0           ['mixed10[0][0]']
 alAveragePooling2D)

 dense (Dense)                  (None, 1024)         2098176     ['global_average_pooling2d[0][0]'
                                                                 ]

 dense_1 (Dense)                (None, 200)          205000      ['dense[0][0]']

==================================================================================================
Total params: 24,105,960
Trainable params: 2,303,176
Non-trainable params: 21,802,784
```

This is the training stage, making an input in the neural network - 61% was the final result

```
Entrée [6]: tensorboard = TensorBoard(log_dir='./logs')

            model.fit_generator(train_generator, steps_per_epoch=32, epochs=100, callbacks=[tensorboard], verbose=2)
```

```
Epoch 92/100
32/32 - 83s - loss: 1.3220 - accuracy: 0.6279 - 83s/epoch - 3s/step
Epoch 93/100
32/32 - 83s - loss: 1.3698 - accuracy: 0.6172 - 83s/epoch - 3s/step
Epoch 94/100
32/32 - 89s - loss: 1.3628 - accuracy: 0.6123 - 89s/epoch - 3s/step
Epoch 95/100
32/32 - 87s - loss: 1.3205 - accuracy: 0.6182 - 87s/epoch - 3s/step
Epoch 96/100
32/32 - 88s - loss: 1.2083 - accuracy: 0.6592 - 88s/epoch - 3s/step
Epoch 97/100
32/32 - 85s - loss: 1.3534 - accuracy: 0.6221 - 85s/epoch - 3s/step
Epoch 98/100
32/32 - 79s - loss: 1.2929 - accuracy: 0.6367 - 79s/epoch - 2s/step
Epoch 99/100
32/32 - 84s - loss: 1.3349 - accuracy: 0.6230 - 84s/epoch - 3s/step
Epoch 100/100
32/32 - 86s - loss: 1.3271 - accuracy: 0.6172 - 86s/epoch - 3s/step
```

```
Out[6]: <keras.callbacks.History at 0x1d09f52e7c0>
```

Evaluate generator - is the evaluation of the previously executed training, despite the warnings, it went well, as it performed the 5 thousand steps and found 64% accuracy.

```
: print(model.evaluate_generator(test_generator, steps=5000))
```

```
C:\Users\FernandaAlmeidaRuas\AppData\Local\Temp\ipykernel_107968\2476778543.py:1: UserWarning: `Model.evaluate_generator` is de
precated and will be removed in a future version. Please use `Model.evaluate`, which supports generators.
  print(model.evaluate_generator(test_generator, steps=5000))
```

```
WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at
least `steps_per_epoch * epochs` batches (in this case, 5000 batches). You may need to use the repeat() function when building
your dataset.
[1.242915391921997, 0.6441296339035034]
```

Finally, we carried out the complete training (true) with the trained cnn network + FC of the 200 images - In order to evaluate the percentage of network learning. We got 81%

```
# unfreeze all layers for more training
for layer in model.layers:
    layer.trainable = True

# we need to recompile the model for these modifications to take effect
# we use SGD with a low learning rate
model.compile(optimizer=SGD(lr=0.0001, momentum=0.9), loss='categorical_crossentropy', metrics=['accuracy'])

model.fit_generator(train_generator, steps_per_epoch=32, epochs=100)
```

```
Epoch 98/100
32/32 [==============================] - 279s 9s/step - loss: 0.5577 - accuracy: 0.8408
Epoch 99/100
32/32 [==============================] - 278s 9s/step - loss: 0.6061 - accuracy: 0.8193
Epoch 100/100
32/32 [==============================] - 277s 9s/step - loss: 0.6217 - accuracy: 0.8174
```

Learning test with 88% accuracy

```
test_generator.reset()
print(model.evaluate_generator(test_generator, steps=5000))
```

```
C:\Users\FernandaAlmeidaRuas\AppData\Local\Temp\ipykernel_107968\2534557482.py:2: UserWarning: `Model.evaluate_generator` is de
precated and will be removed in a future version. Please use `Model.evaluate`, which supports generators.
  print(model.evaluate_generator(test_generator, steps=5000))
```

```
WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at
least `steps_per_epoch * epochs` batches (in this case, 5000 batches). You may need to use the repeat() function when building
your dataset.
[0.4003739356994629, 0.8839497566223145]
```

CNN network storage to use in the next step

```
model.save("birds-inceptionv3.model")
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolu
tion_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing 5 of 94). These functions will not be
directly callable after loading.
```

```
INFO:tensorflow:Assets written to: birds-inceptionv3.model\assets
```

```
INFO:tensorflow:Assets written to: birds-inceptionv3.model\assets
```

Predict

- **Let's do a test, using the model of the already trained CNN network.**

- **Our goal is to see how much the CNN network is able to predict with images: We choose an image of a specific bird and ask the network to verify.**

1- Step was to load the list of types of birds from the images directory - including in the already trained CNN network

```
CLASS_NAMES = sorted(listdir('images'))

model = load_model('birds-inceptionv3.model')
```

2- Creation of image reading and formatting function to present the CNN network - for prediction

```
def predict(fname):
    img = image.load_img(fname, target_size=(ROWS, COLS))
    img_tensor = image.img_to_array(img) # (height, width, channels)
    # (1, height, width, channels), add a dimension because the model expects this shape:
    # (batch_size, height, width, channels)
    img_tensor = np.expand_dims(img_tensor, axis=0)
    img_tensor /= 255. # model expects values in the range [0, 1]
    prediction = model.predict(img_tensor)[0]
    best_score_index = np.argmax(prediction)
    bird = CLASS_NAMES[best_score_index] # retrieve original class name
    print("Prediction: %s (%.2f%%)" % (bird, 100*prediction[best_score_index]))
```

The CNN network predicted with 97.32% that the image was a bird of the species - Annas`s Hummingbird

```
predict('annas_hummingbird_sim_1.jpg')
```

```
1/1 [==============================] - 11s 11s/step
Prediction: 067.Anna_Hummingbird (97.32%)
```

In the second test, we were not successful, we believe it was due to the choice of image.

```
# interactive user input
while True:
    fname = input("Enter filename: ")
    if(len(fname) > 0):
        try:
            predict(fname)
        except Exception as e:
            print("Error loading image: %s" % e)
    else:
        break
```

```
Enter filename: House_wren.jpg
1/1 [==============================] - 0s 85ms/step
Prediction: 190.Red_cockaded_Woodpecker (25.21%)
```

5. Evaluation and feedback

In this project, we reproduced two different tasks: the first one was "Identifying handwritten mathematical symbols with CNNs," and the second was "Revisiting the bird species identifier to use images" Through both exercises, we gained an understanding of the basics of deep learning.

We learned that a large amount of data is necessary for achieving better results. Additionally, a powerful computer is essential for running all the data and codes. This is why we had to limit the number of images we used from the dataset in the first exercise.

Although some of our results were different from those presented in the book, we learned that the first step in a deep learning project is to import and prepare the available data, ensuring that the formats and information are most efficient and recommended for a neural network.

We also learned that the next step is to train the machine to access predictions finally.

Since we were not familiar with Python, deep learning, and these concepts, reproducing everything perfectly was challenging for us. We used external sources to help us achieve the results presented in this project.

In conclusion, we all agree that working on a project in a field where we have no background can be a way to learn new concepts. However, we believe that if we could learn these concepts in class, the outcome could be even better.

6. Conclusion

Deep learning, which is part of machine learning and a broader field of artificial intelligence, is a powerful tool for predicting results, as we have seen in these projects. The process involves two main parts: first, creating a model by training a machine, and second, utilizing the model that has been created.

Contrary to what some people may believe, deep learning is not only useful for business or leisure purposes. In the field of healthcare, deep learning is evolving, and in the not-so-distant future, several apps and software could be available to predict illnesses, for example, by analyzing images sent by a user. Another important application of deep learning can be in earthquake prediction.

The business world can also benefit from the application of deep learning, including entertainment companies, as we see with Netflix. By using a large amount of data, the company can predict users' preferences and suggest series and movies more likable to be watched, increasing their engagement levels. Advertising is another example that can benefit from deep learning and its predictive capacity.

Consequently, deep learning will probably be an essential tool, driving innovation in a wide range of industries.

7. Appendix: software and datasets

- Dataset HASYv2 - https://www.kaggle.com/datasets/guru001/hasyv2
- Software Kaggle - https://www.kaggle.com/
- Jupyter notebook - https://jupyter.org/
- Keras - https://stackoverflow.com/questions/56641165/modulenotfounderror-no-module-named-keras-for-jupyter-notebook

References

An Interactive Node-Link Visualization of Convolutional Neural Networks. (s. d.). https://adamharley.com/nn_vis/
Stewart, M., PhD. (2021, 8 décembre). Simple Introduction to Convolutional Neural Networks. Medium. https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac
Nazemi, A., Tavakolian, N., Fitzpatrick, D., & Suen, C. (2019, 16 octobre). Offline handwritten mathematical symbol recognition utilising deep learning. ResearchGate. https://www.researchgate.net/publication/336602577_Offline_handwritten_mathematical_symbol_recognition_utilising_deep_learning
Introduction:
What Is Deep Learning? | How It Works, Techniques & Applications - MATLAB & Simulink (mathworks.com). https://www.mathworks.com/discovery/deep-learning.html