|  |  |  |
| --- | --- | --- |
| f |  |  |
| **Benha University** |  | **Faculty of Computers & Artificial Intelligence** |

**Simpsons classification**

**Computer Science Departement:**

***Project Team***

**هبه ماهر الشحات(4)**

**هبه عادل على(4)**

**(4)نورهان ايمن عبدالمعطي**

***Under Supervision of***

**Dr. Ibrahim**

**En. Mahmoud mansour**

Benha, May 2024

ABSTRACT

it's a publicly available dataset of Simpsons characters designed for image classification tasks . It contains pictures featuring various characters from the show, each labeled accordingly . The dataset includes roughly 45,000 images in total, split across 48 characters . While the variety is impressive, the number of images per character can vary significantly, with some having less than 400 . To address this imbalance, researchers often focus on a subset of the most well-represented characters, typically around 18. These characters come in various poses and scenes, sometimes even interacting with others . The dataset's purpose is to aid in the development of machine learning models that can recognize and classify characters within images.

Table of Contents

[List of figures ii](#_Toc222557974)

[list of tables iii](#_Toc222557975)

[LIST OF ACRONYMS/ABBREVIATIONS iv](#_Toc222557976)

[1 CHAPTER ONE: Introduction 1](#_Toc222557977)

[2 CHAPTER TWO: Data processing 2](#_Toc222557977)

[3 CHAPTER THREE: Model Building 5](file:///C:\Users\lenovo\Downloads\Documentation%20Template.docx#_Toc222557977)

[1 CHAPTER FOUR: MOdel EVALUATION 1](file:///C:\Users\lenovo\Downloads\Documentation%20Template.docx#_Toc222557977)

List of figures

[Figure ‎1‑1: Data processing. 2](#_Toc211232407)

[Figure ‎1‑2: Data processing 3](#_Toc211232408)

[Figure ‎1‑3: layers of the model 5](#_Toc211232409)

list of tables

[Table ‎1‑1: summary of the model.](#_Toc211232411) 7

LIST OF ACRONYMS/ABBREVIATIONS

|  |  |
| --- | --- |
| ACRONYM | Definition of Acronym |
|  |  |

Chapter One

# Introduction

Dive into the world of Springfield with a popular dataset dedicated to classifying characters from The Simpsons! This collection of images provides a treasure trove for machine learning enthusiasts working on image recognition tasks.

The dataset boasts a vast collection of around 45,000 images featuring a wide range of characters from the beloved animated sitcom. Each image is meticulously labeled with the corresponding character's name, making it perfect for supervised learning algorithms.

While the dataset offers an impressive character variety with 48 individuals, the distribution of images isn't entirely uniform . Some characters may have a smaller pool of images, with some falling below 400 . To tackle this challenge, researchers often concentrate on a well-represented subset, usually around 18 prominent characters .

The beauty of this dataset lies in the portrayal of characters in various scenarios. You'll find them in different poses, settings, and sometimes even interacting with each other. This diversity adds a layer of complexity that closely mimics real-world scenarios, making it a valuable tool for training robust image classification models.

So, if you're looking to explore the fascinating world of image recognition with a touch of Springfield flair, this Simpsons character classification dataset is an excellent place to start!

**DATASET:**

# We used a dataset from Kaggle named “ The Simpsons Characters Data”

Chapter Two

Dataset acquisition and preprocessing

Data acquisition: is the first part on building your model journey as you cannot train your model without data.

The dataset in our case should be images for all the different classes of the cards game which are 43 different classes. We actually find and use a dataset from Kaggle platform, which contains thousands of datasets for a wide area of different tasks such as (NLP, image classification, regression, …, etc.).

**Data preprocessing:**

Data pre-processing is crucially important to a model’s performance.

After we acquisitor our data, we should ensure that our data is good to train a prefect model that mostly predicts the answers correctly as possible, to ensure that:

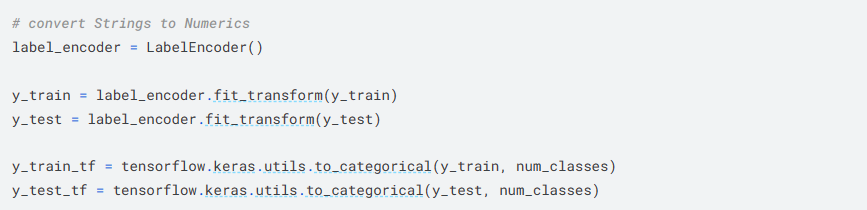


Figure ‎2‑1: data processing.

 **Label Encoding:**

* label\_encoder = LabelEncoder(): This line imports and creates a LabelEncoder object from the sklearn.preprocessing library. This encoder is used to convert textual labels (like character names) into numerical values.
* y\_train = label\_encoder.fit\_transform(y\_train): This line fits the LabelEncoder to the training labels (y\_train). During fitting, the encoder learns the unique categories (character names) present in the data and assigns a unique integer value to each. It then transforms the training labels by replacing each category with its corresponding integer value. This creates a numerical representation of the labels suitable for machine learning models.
* y\_test = label\_encoder.transform(y\_test): This line transforms the test labels (y\_test) using the already fitted LabelEncoder. Since the encoder has already learned the mapping from categories to integers during training, it can directly apply that mapping to the unseen test labels.

 **One-Hot Encoding (Optional):**

* tensorflow.keras.utils.to\_categorical(y\_train, num\_classes): This line (if used) performs one-hot encoding on the training labels (y\_train). One-hot encoding creates a vector with the same length as the number of categories (characters). Each position in the vector is set to 0, except for the index corresponding to the category of the data point, which is set to 1. This is a common way to represent categorical features in neural network models.
* y\_test\_tf = tensorflow.keras.utils.to\_categorical(y\_test, num\_classes): Similar to the training labels, this line (if used) applies one-hot encoding to the test labels (y\_test).
* num\_classes: This argument (which might be defined elsewhere) specifies the number of unique character categories (likely 48 for the Simpsons dataset).



Figure ‎2‑2: Data processing.

1. **Import and File Copying:**
   * from distutils.file\_util import copy\_file: This line imports the copy\_file function from the distutils.file\_util module. This function is used to copy a file from one location to another.
   * The code then defines two variables:
     + fromDirectory: This points to the source file location, likely containing the initial model weights (../input/the-simpsons-characters-dataset/weights.best.hdf5).
     + toDirectory: This specifies the destination for the copied file (weights.best.hdf5).
   * Finally, copy\_file(fromDirectory, toDirectory) copies the file from the source to the destination. This might be done to ensure a starting point for the model training.
2. **Saving the Best Model:**
   * checkpointer = ModelCheckpoint(filepath="weights.best.keras", verbose=0, save\_best\_only=True): This line creates a ModelCheckpoint object, a common tool in Keras (a deep learning library) used for saving models during training.
     + filepath="weights.best.keras": This defines the filename pattern for the saved models. Here, it will be "weights.best.keras" with an incrementing number appended if multiple are saved.
     + verbose=0: This disables printing information every time a model is saved.
     + save\_best\_only=True: This tells the ModelCheckpoint to only save the model that achieves the best validation performance (metric not explicitly mentioned here). Essentially, it keeps track of the best model seen so far.
3. **Model Training Parameters:**
   * input\_shape = (img\_size, img\_size, 3): This defines the expected input shape for the model. It likely takes images with a width of img\_size pixels, a height of img\_size pixels, and 3 channels (RGB color).
   * learning\_rate = 0.001: This sets the learning rate, a hyperparameter that controls how much the model updates its weights during training.
   * weight\_decay = 1e-4: This defines the weight decay, a technique that penalizes large weights during training, helping to prevent overfitting (the model memorizing the training data instead of learning general patterns).

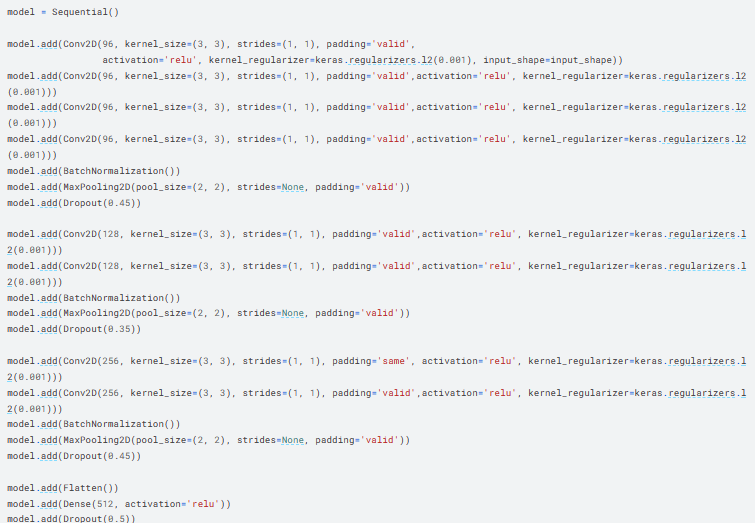
In summary, this code snippet prepares for training an image classification model on the Simpsons characters dataset. It copies an initial set of weights (potentially) and sets up a mechanism to save only the best performing model during training based on validation performance.

Chapter Three

# Model Building

Our model consists of :

* **Convolution layers**: 12
* **Max Pooling layers**: 3
* **Flatten layer**: 1
* **Dense layers**: 3
* **Batch Normalization layers**:3
* **Dropout layers:4**



A screen shot of a computer

Description automatically generated

There are a total of **12 Convolution layers (Conv2D)** identified in the code.

* Four repetitions of Conv2D with 96 filters each.
* Two repetitions of Conv2D with 128 filters each.
* Two repetitions of Conv2D with 256 filters each.

1. **Extracting Features:**
   * Each Conv2D layer applies a set of learnable filters (small matrices) to the input image (or the output of the previous layer).
   * As the filter slides across the image, it performs element-wise multiplication between its weights and the corresponding elements in the image patch. The results are then summed up to create a single value for the output feature map.
   * This process essentially helps the layer identify specific patterns or features within the image, like edges, lines, or shapes.
2. **Learning Specific Features:**
   * There are multiple filters within each Conv2D layer (96, 128, or 256 in this case). Each filter learns to detect a different kind of feature.
   * For example, some filters might specialize in detecting vertical edges, while others might focus on horizontal lines or corners.
3. **Creating Feature Maps:**
   * The output of a Conv2D layer is a collection of 2D feature maps, where the width and height are reduced compared to the input (due to strides and padding choices).
   * Each feature map captures the presence of a specific feature learned by a particular filter across the entire image.

**Increasing Complexity Through the Network:**

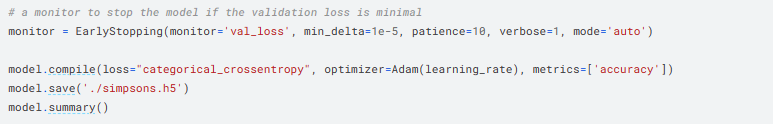
* The initial Conv2D layers (with 96 filters) likely focus on capturing basic features like edges and simple shapes.
* As we move through the network (to layers with 128 and 256 filters), the complexity of the learned features increases. These layers might start to combine simpler features to detect more intricate patterns or objects within the characters' images.

**Additional Points:**

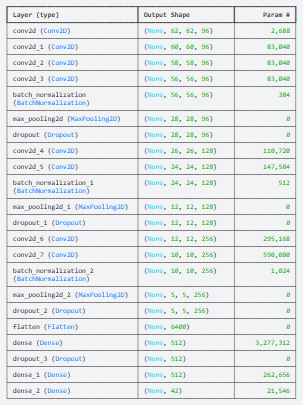
* The specific features each Conv2D layer learns depend on the training data (Simpsons characters in this case) and the overall network architecture.
* Techniques like ReLU activation (used here) introduce non-linearity, allowing the network to learn more complex relationships between features.

While each Conv2D layer performs a similar function, the combination of these layers with pooling, normalization, and dropout helps the network progressively extract and combine features to achieve accurate character classification.

This code provides early stopping:



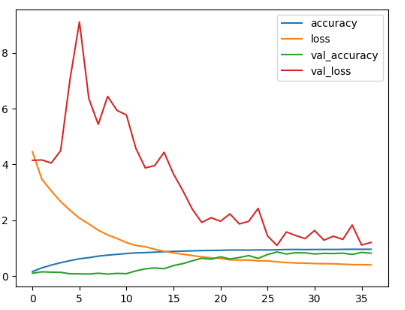
And this is the summary of the model:

* 
* **Total params:** 4,958,794 (18.92 MB)
* **Trainable params:** 4,957,834 (18.91 MB)
* **Non-trainable params:** 960 (3.75 KB)

Code for training model :



**This diagram for accuracy ,val\_acc,loss and val\_locc:**



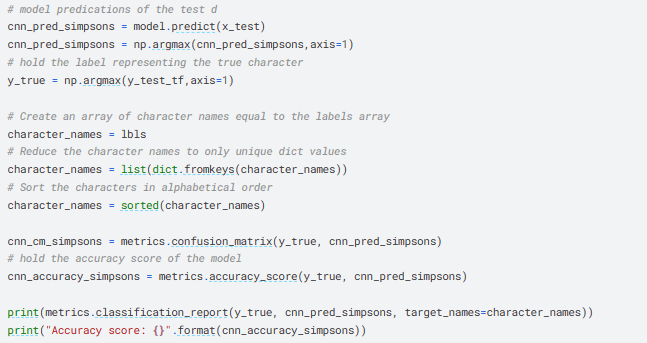
**Chapter 4: Model evaluation**

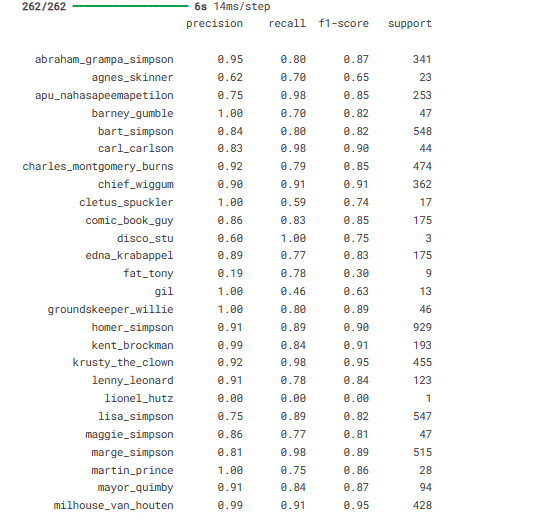
various evaluation metrics can be employed depending on the specific objectives and requirements of the task. Some commonly used metrics include:

1. Accuracy: It measures the overall correctness of the model's predictions It is calculated as the ratio of correctly classified samples to the total number of samples.
2. Precision: Precision is the proportion of true positive predictions (correctly identified diseased plants) out of all positive predictions (all predicted diseased plants). It indicates the model's ability to accurately detect diseased plants. Precision = TP / (TP + FP).
3. Recall (Sensitivity): Recall represents the proportion of true positive predictions out of all actual positive samples (all diseased plants). It measures the model's ability to identify diseased plants correctly.

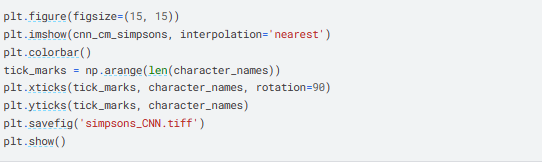
Recall = TP / (TP + FN)

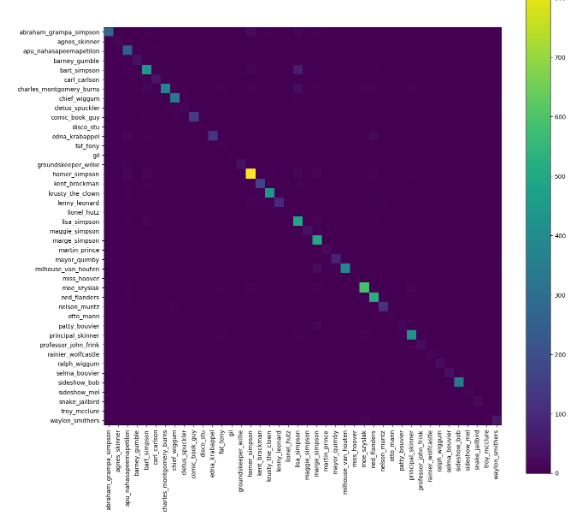
1. F1 Score: The F1 score is the harmonic mean of precision and recall. It provides a balanced measure that takes into account both precision and recall. It is often used when there is an imbalance between the number of diseased and healthy plants in the dataset.





**confusion matrix:**

****

****

**GUI:**

****

* Select image button : to select image from gallery.
* Make prediction: to get the predict of the model