Università degli Studi di Milano
Data Science and Economics (DSE)

PROJECT 4: FACE/COMIC RECOGNIZER

Fatima Otegen
Email: fatima.otegen@studenti.unimi.it, 971106

Introduction. This paper presents an empirical analysis of the performance of popular convolutional neural networks (CNNs) for classifying two classes either Real Face pictures or Comic pictures. The main goal of this experimental project is using Deep Learning algorithms to teach a computer how to identify image categories. **Neural nets** are a means of doing machine learning, in which a computer learns to perform some task by analyzing training examples. It consists of an input layer**,** one or more hidden layers, and an output layer *(MIT's definition)*. Using various Deep Learning Algorithms, they can identify these hidden patterns and correlations in raw data, cluster and classify them. The task of assigning one label from a fixed set of categories to an input image is known as image classification. This is one of the most important problems in computer vision, and despite its simplicity, it has a wide range of applications.

Problem Statement. For this experimental project, I am going to use the Comic Faces dataset that consists of a number of images and the given images are collected from 2 classes such as *comics, faces*

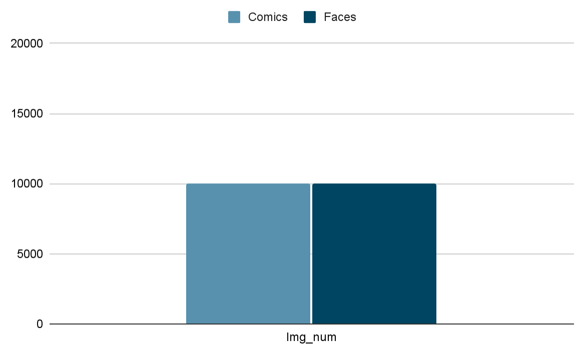


Figure 1: Number of images

To further understand, let's look at an example. Our system will accept an image as input for performing image categorization, such as a Comic image. The system will now be aware of a collection of categories, with the purpose of assigning a category to each image. This problem may appear simple or straightforward, yet it is really difficult for a computer to solve. As you may know, the computer sees a grid of numbers rather than a humorous image as we do. Images are three-dimensional arrays of integers ranging from 0 to 255 in size, with dimensions of Width x Height x 3. The three color channels, Red, Green, and Blue, are represented by the number three.

In this given example each image sample is labeled with the class to which it belongs. The dataset contains 20000 images with 1024x1024 size.

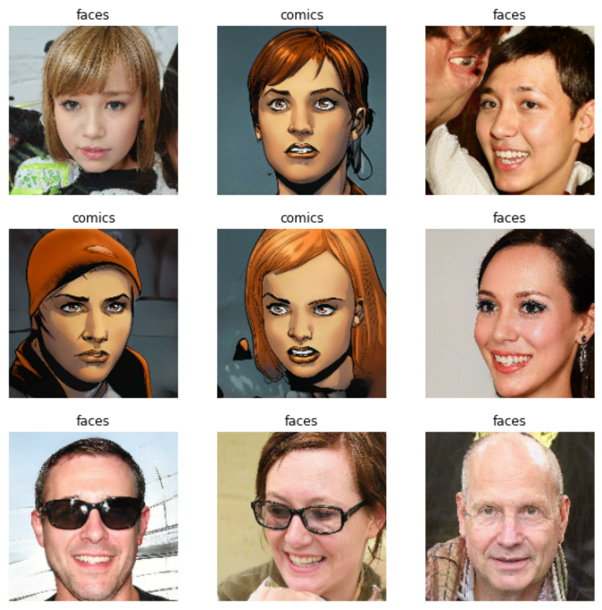


Figure 2: Sample of images from dataset

Data Preparation. It can be difficult to determine exactly how to prepare images for modeling, such as scaling or normalizing pixel values, when using convolutional neural networks for image categorization. First in order to facilitate mini-batch learning, we need to have a fixed shape for the images inside a given batch. This is why an initial resizing is required. Principally, our machine learning models train faster on smaller images, however our initial size was 1024x1024, therefore I set up the image batch as 64 and the shape 180X180. Eventually, we have a tensor of the shape (32, 180, 180, 3). This is a batch of 32 images of shape 180x180x3 (the last dimension refers to color channels RGB).

To make sure to use buffered prefetching so we can yield data from disk without having I/O become blocking. These are two important methods that I have used when loading data: Dataset.cache keeps the images in memory after they're loaded off disk during the first epoch. This will ensure the dataset does not become a bottleneck while training the model. Dataset.prefetch overlaps data preprocessing and model execution while training. The RGB channel values are between [0, 255] range. But this isn't ideal for a neural network, since we should aim for small input values in general. Using tf.keras.layers.rescaling we've normalized values to be in the [0, 1] range. Moreover, to avoid overfitting problems on the model I applied Data Augmentation where we increase the amount of training data using information only in our training data by adding random (but realistic) transformations, such as Image zoom, Image rotation, Image flip.

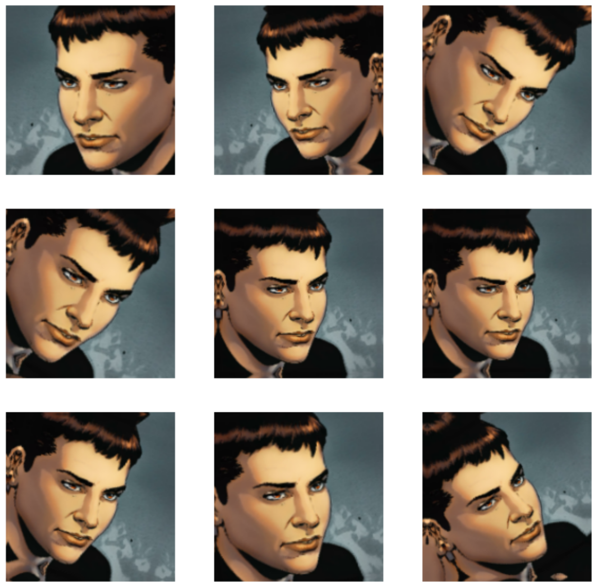
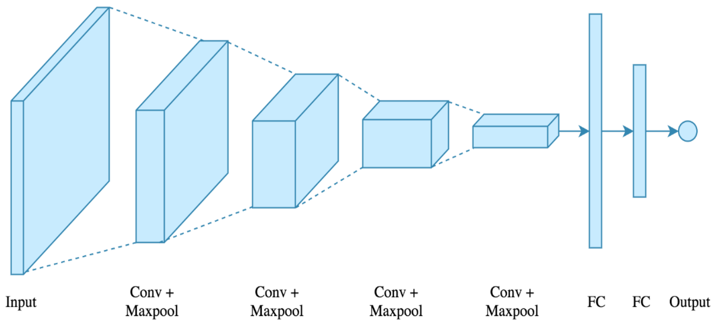


Figure 3: Data Augmentation

Dataset is splitted by train and validation part, using 80% of the images for training, and 20% for validation.

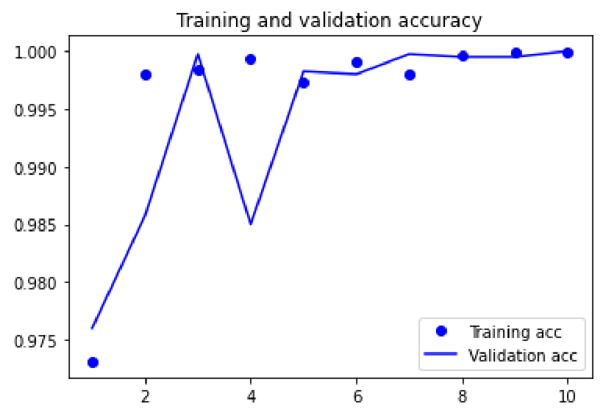
Building the model. In this experimental project I have used Convolution Neural Network(CNN), which is a class of Neural Network that has proven very effective in areas of image recognition, processing, and classification. The CNN model requires training data in order to calculate training weights and validation in order to assess its performance. Each input image is processed through a convolution layers that include Kernels as illustrated in the image below:



A fully connected layer functions as a classifier, while the Convolution + MaxPooling layers retrieve features from the input image. In the example above, when the network receives an image as input, it assigns the highest probability to it and predicts which class the image belongs to. In addition, the type of padding used here is “same padding”, which literally means the output image is equal to the input image. For the Pooling Layer, Max pooling was applied that extracts only those features which have the highest value. Finally, In Fully Connected Layer -each node is connected to every other node in the adjacent layer and uses the sigmoid function for the commuting class of input image. To sum up, our *Sequential* model consists of three convolution blocks (tf.keras.layers.Conv2D) with a *max pooling layer* (tf.keras.layers.MaxPooling2D) in each of them. There's a fully-connected layer (tf.keras.layers.Dense) with 128 units on top of it that is activated by a ReLU activation function ('relu'). Furthermore, before training the model with augmented images another technique to reduce overfitting ‘Dropout’ regularization have used to the network. When we apply Dropout to a layer, it randomly drops out (by setting the activation to zero) a number of output units from the layer during the training process. Dropout takes a fractional number as its input value, in the form such as 0.1, 0.2, 0.4, etc. This means dropping out 10%, 20% or 40% of the output units randomly from the applied layer.

Train the model. Builded model was trained by SparseCategoricalCrossentropy loss and with 'adam' optimizer, which is the updated version of RMSProp. According to Kingma et al., 2014, the 'adam' method is *"computationally efficient, has little memory requirement, invariant to diagonal rescaling of gradients, and is well suited for problems that are large in terms of data/parameters"*. The model was trained for 10 epochs and to complete the training it took about 1,5 hours.

Results. In the result, we check performance on both the training and validation sets (evaluation metric is accuracy). The training accuracy comes out to be 99,6% and the validation accuracy is 99,4%



As we can see from above illustrated line chart for 10 epochs we had almost similar accuracy with a slight difference in 4th epoch between training and validation accuracy about 0.015% .

When we predicted using random comic and real face images our model forecasted categories with high accuracy of 99.99%.