# Image Classification with Convolutional Neural Networks

## (Keras and/or TensorFlow)

## Description:

In this task, it is being assigned to use the Convolutional Neural Network using Keras or TensorFlow. Some common or widely used datasets such as MNIST, CIFAR-10, CIFAR-100 or MNIST-like and CIFAR-like are removed from the list. Therefore, for the solution we are using the Animal classification dataset. Where the dataset contains a fixed and balanced number of 5 classes, which includes “CAT”, “DOG”. “ELEPHANT”, “HORSE” and “LION”.

To develop a smart model of Artificial Intelligence that could classify test images, we are using the CNN (Convolutional Neural Network) algorithm. The algorithm that is used wherever there is a need of processing images and then to extract relevant information out of it.

The process of Image classification has been divided into several parts which we will explore in the next heading: Processing and Analysis.

## Processing and Analysis

In this section we will be covering the practical implementation that took place to build of Image processing model.

### Pre-processing

The section of data preprocessing can be considered as one of the main pillars, if someone wants to achieve a good performing result. This part of preprocessing is divided into several steps, mentioned below:

1. **Step1:** Splitting the dataset into train, test, and validation part. Often, it is considered as a healthy step where the dataset is divided into 3 different balanced part. The common practice to divide the folders into train, test, and validation is 70:10:20 percent.
2. **Step 2:** The first and foremost task is to check out whether the dataset is balanced or imbalanced. In our case the dataset turns out to be balanced but if the someone faces the situation of an imbalanced dataset in image processing, one useful technique is to use the tool of data augmentation in Keras to generate more required images from the dataset by applying different ways such as: adding shears, rotating, blurring etc
3. **Step 3:** Second step is to rescale the images training, testing and validation images. The need to rescale images to a fixed size between 0 and 1 is to reduce the processing time and to avoid using of larger weights. One another aspect to understand rescales is that, if we have the fixed size in a larger number, therefore, the shirking of images will be less and less shrinking eventually leads to less deformation of objects features and their patterns which resides within the image.

### Data Modeling

Building any machine learning or deep learning model is the engine of any Ai project. In this specific task, it is guided to used 3 different Deep Learning CNN model, hyper-parameter tunings:

1. **Model 1:** The hidden topology of the fist model contains 4 convolutional 2d layers, 2 MaxPooling, 2 BatchNormalization and 1 dense layer preceding with a dropout layer. The structure of this model is as follow.

Sequential Architecture

Input Layer 🡪 128x128x3 size

Convolution 2d layer 🡪 32 neurons, 3x3 filter, activation: ReLU

Convolution 2d layer 🡪 32 neurons, 3x3 filter, activation: ReLU

MaxPooling 2d 🡪 2x2

BatchNormalization 🡪 Default parameters

Convolution 2d layer 🡪 64 neurons, 3x3 filter, activation: ReLU

Convolution 2d layer 🡪 64 neurons, 3x3 filter, activation: ReLU

MaxPooling 2d 🡪 2x2

BatchNormalization 🡪 Default parameters

Flatten Layer 🡪 Default parameters

Dense Layer 🡪 128 neurons, activation: ReLU

Dropout Layer 🡪 0.50

Output Layer 🡪 5 neurons, activation = softmax

After that, we then used the Adam Optimizer having learning rate of 0.001, metrics as accuracy And loss as categorical\_crossentropy. In the model1.fit section, we are using the value of steps\_per\_epoch as 32, epochs as 50 and validation steps as 50.

1. **Model 2:** The hidden topology of the second model contains 4 convolutional 2d layers, 4 MaxPooling, 3 BatchNormalization and 2 dense layer preceding with a dropout layer. The structure of this model is as follow.

Sequential Architecture

Input Layer 🡪 128x128x3 size

Convolution 2d layer 🡪 32 neurons, 3x3 filter, activation: ReLU

MaxPooling 2d 🡪 2x2

Convolution 2d layer 🡪 32 neurons, 3x3 filter, activation: ReLU

MaxPooling 2d 🡪 2x2

BatchNormalization 🡪 Default parameters

Convolution 2d layer 🡪 64 neurons, 3x3 filter, activation: ReLU

MaxPooling 2d 🡪 2x2

Convolution 2d layer 🡪 64 neurons, 3x3 filter, activation: ReLU

MaxPooling 2d 🡪 2x2

BatchNormalization 🡪 Default parameters

Flatten Layer 🡪 Default parameters

Dropout Layer 🡪 0.50

Dense Layer 🡪 32 neurons, kermel\_regularizer: l2(0.01), bian\_regularizer: l2(0.01)

Dense Layer 🡪128 neurons, activation = ReLU.

Output Layer 🡪 5 neurons, activation = softmax

After that, we then used the SGD(sophisticated Gradient Descend) Optimizer having learning rate as default, metrics as accuracy And loss as categorical\_crossentropy. In the model2.fit section, we are using the value of steps\_per\_epoch as 64, epochs as 50 and validation steps as 70.

1. **Model 3:** The hidden topology of the second model contains 4 convolutional 2d layers, 4 MaxPooling, 4 BatchNormalization and 2 dense layer preceding with a dropout layer. The structure of this model is as follow.

Sequential Architecture

Input Layer 🡪 128x128x3 size

Convolution 2d layer 🡪 128 neurons, 5x5 filter, activation: ReLU

MaxPooling 2d 🡪 2x2 pool size

BatchNormalization 🡪 Default parameters

Convolution 2d layer 🡪 64 neurons, 3x3 filter, activation: ReLU

MaxPooling 2d 🡪 2x2 pool size

BatchNormalization 🡪 Default parameters

Convolution 2d layer 🡪 32 neurons, 3x3 filter, activation: ReLU

MaxPooling 2d 🡪 2x2 pool size

BatchNormalization 🡪 Default parameters

Convolution 2d layer 🡪 32 neurons, 3x3 filter, activation: ReLU

MaxPooling 2d 🡪 2x2 pool size

BatchNormalization 🡪 Default parameters

Flatten Layer 🡪 Default parameters

Dropout Layer 🡪 0.50

Dense Layer 🡪 512 neurons, kermel\_regularizer: l2(0.01), bian\_regularizer: l2(0.01)

Output Layer 🡪 5 neurons, activation = softmax

After that, we then used the RMSprop Optimizer having learning rate as 0.004, metrics as accuracy And loss as categorical\_crossentropy. In the model3.fit section, we are using the value of steps\_per\_epoch as 128, epochs as 50 and validation steps as 70.

### Results

After building the models, the next step is to train them on the dataset (train set) and evaluate those results. The results are being noticed in each of the models. Where similar number of epochs have been used, which is 50. Below are the results of each model after the 50tth epoch.

Shape, rectangle

Description automatically generated

Fig 1. Train and validation loss and accuracy for model 1

Shape, rectangle

Description automatically generated

Fig 2. Train and validation, loss and accuracy for model 2

Shape

Description automatically generated

Fig 3. Train and validation, loss and accuracy for model 3

In the last and 50th epoch of every model. We can notice that the train accuracy in fig1 and fig2 are quite the similar, whilst in the 3rd model, the accuracy got enhanced.

In the work of deep learning, one thing is quite certain is, whenever we come across any image processing task, it is quite certain that the model training part is quite rigorous. Therefore, excessive training is important wherever there is a concern to increase the performance or to check the limits of the any deep learning model. In the Fig3, we the notice that the 3rd model performed better at the 50th epoch. Where the validation accuracy is 81% and train accuracy is 96%. Which is certainly can be considered as a good model if we train it more.

### CONCLUSION

“There is always a room for improvement”

The above line fits very well for our problem statement. Where, it is quite common within the Ai community to run, train, test and repeat the process. Improvement is important but keeping the room empty is also crucial because perfection cannot be achieved and anything today which is perfect is biased. Therefore, we should need to follow the rule of being generic in deep learning to achieve good results.