ANIMAL CLASSIFICATION

Using DEEP LERNING

IT IS TO IDENTIFY ANIMALS.

BY

Prabhashi Mahawatta
(COHNDDS(F/T)21.1F-017)

TABLE OF CONTENTS

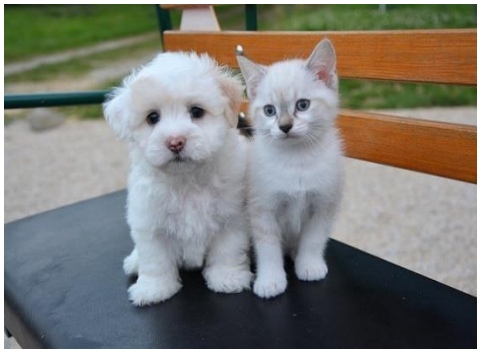
Abstract

A real-world animal biometric system which detects and describes animal life in image and

video is an emerging subject in machine vision. These systems provide computer vision approaches for the classification of animals. A novel method for animal face classification based on one of the popular convolutional neural network (CNN) features. I am using CNN in this project which can automatically extract features, learn and classify them. The proposed method can also be used in other areas of image classification and object recognition.

Key Words: Animal classes, CNN.

1. INTRODUCTION



Animal recognition and classification is an important area which has not been discussed rapidly. Animal classification which relies on the problem of distinguishing images of different animal species is an easy task for humans, but evidence suggests that even in simple cases like cats and dogs, it is difficult to distinguish them automatically.

Animals have flexible structure that could self-mask and usually they appear in complex scene. Also, as all objects, they may appear under different illumination conditions, viewpoints and scales. There are attempts to apply recognition methods on images of animals but the specific problem of animal categorization has recently attracted limited interest. Many existing methods showing promising results for human face recognition cannot properly represent the diversity of animal classes with complex intraclass variability and interclass similarity. There are several kinds of approaches for solving this problem with each one having its advantages and disadvantages. The first approach constructs complex features which represents and discriminates sample images better but creating such a feature is complicated and it is problem dependent. The second approach combines the extracted features from different methods and concatenates them to

build a more powerful feature vector. Increasing the size of feature space causes increased problem computation cost. Instead of using complex representation, the information is consolidated from different classifiers and a decision is made according to it. This method is known as score-level fusion. Observing wild animals in their natural environments is an important task in ecology. The fast growth of human population and the endless pursuit of economic development are making over exploitation of natural resources, causing rapid and novel and substantial changes to Earth's ecosystems. An increasing area of land surface has been transformed by human action, altering wildlife population, habitat and behavior.

More seriously, many wild species on Earth have been driven to extinction and many species are introduced into new areas where they can disturb both natural and human systems. Monitoring wild animals, therefore, is essential as it provides researchers evidences to help make conservation and management decisions to maintain diverse, balanced and sustainable ecosystems in the face of those changes, Various modern technologies have been developed for wild animal monitoring, including radio tracking, wireless sensor network tracking, satellite and global positioning system (GPS) tracking, and monitoring by motion sensitive camera traps. Motion-triggered remote cameras or

"camera traps" are an increasingly popular tool for wildlife monitoring, due to their novel features equipped, wider commercial availability, and the ease of deployment and operation. For instance, a typical covert camera model is capable of not only capturing high definition images in both day and night, but also collecting information of time, temperature and moon phase integrated in image data. In addition, flexible camera settings allow tracking animals secretly and continuously.

1.1. VISION

The project's goal is to create an animal image classifier that could achieve the desired accuracy in dense forest environments, and to help ecologists and researchers in neural network/artificial intelligence & zoological domains to further study and/or improve habitat, environmental, and extinction pat.

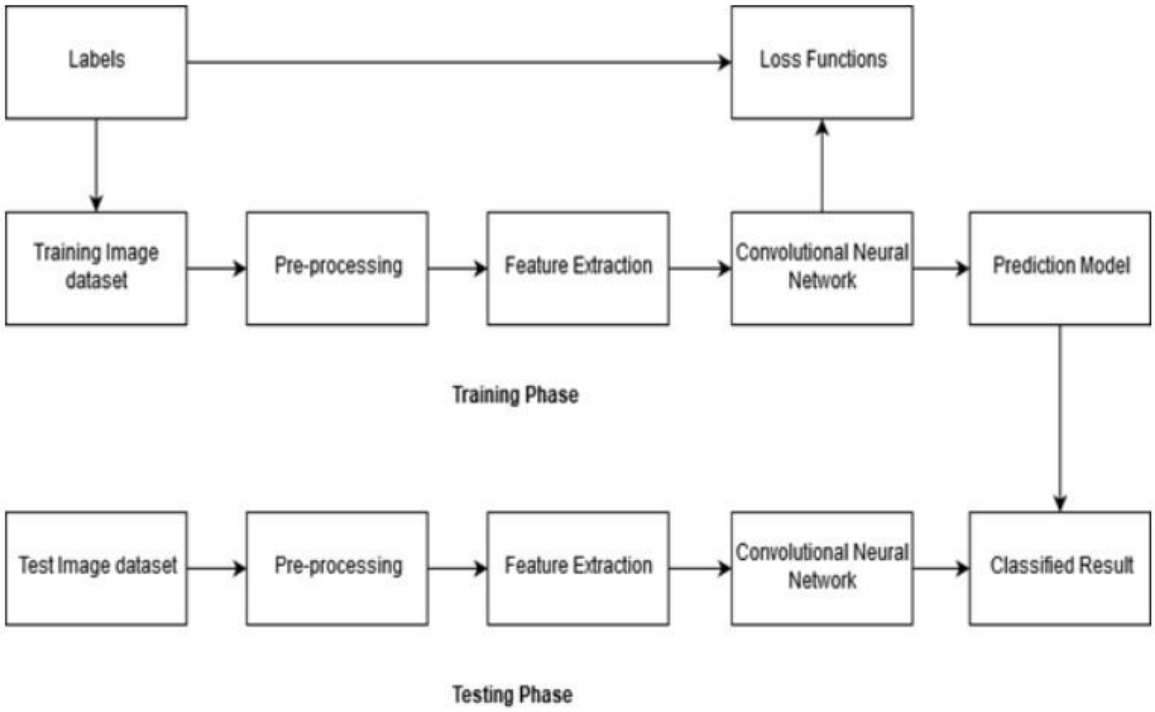
Objectives –

Ø Create a CNN model to find similar features in images belonging to different classes and using them to identify and label images. (Classification of animal images)

Ø Evaluate the loss and accuracy of the model.

2. METHOD SPECIFICATION

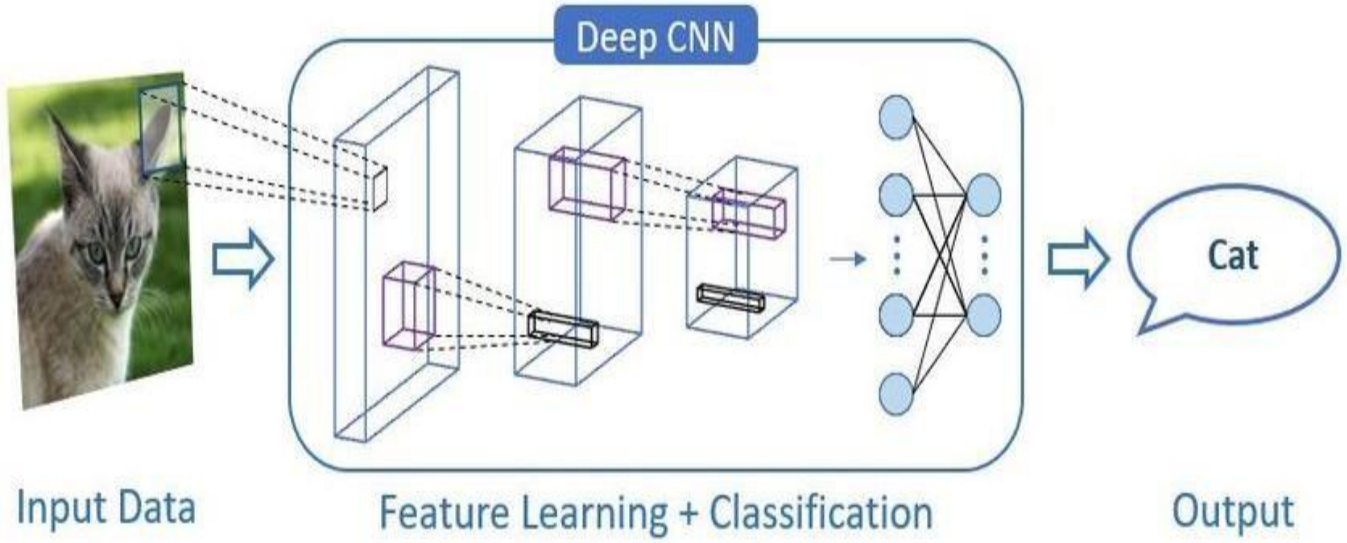
A system architecture diagram is used to show the relationship between different components. Usually they are created for systems which include both hardware and software and these are represented using diagram to show the interaction between them. A novel method is proposed for animal face classification based on one of the popular convolutional neural network (CNN) features. We are using CNN, a method which can automatically extract features, learn and classify them. CNNs relatively does little pre- processing compared to other image classification algorithms. This means that the network learns the filters which were hand-engineered in previous traditional algorithms. This independence from prior knowledge and human effort in feature design is a major advantage in image classification



As shown in the above, the following figure demonstrates the architecture of this model.

2.1. CONCEPT DETAILS.

This model is built by using Convolutional neural network. Convolutional neural networks (CNN) are a special architecture of artificial neural networks, proposed by Yann LeCun in 1988. CNN uses some of the features of the visual cortex. Now that the pre-processing is done, neural network can be implemented. It will produce 3 convolution layers with 2 x 2 max-pooling. Artificial neural networks are one of the main tools used in machine learning. As the name suggests, they are brain-inspired systems which are intended to replicate the way that we humans learn. Neural networks consist of input and output layers and a hidden layer consisting of units that transform the input into something that the output layer can use in most cases. They are excellent tools for finding patterns which are far too complex for a human programmer to extract and teach the machine to recognize. While neural networks have been around since the 1940s, it is only in the last several decades where they have become a major part of artificial intelligence. This is due to the arrival of a technique called "back propagation," which allows networks to adjust their hidden layers of neurons in situations where the output does not match from what the creator is expecting — like a network designed to recognize dogs, which misidentifies a cat, for example.



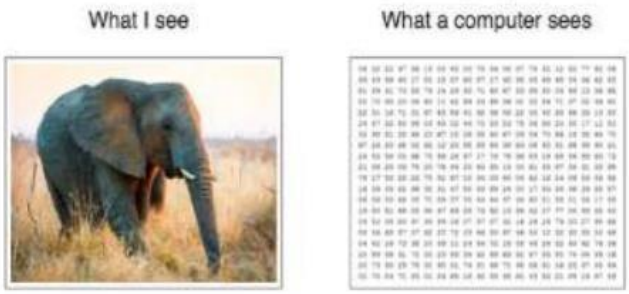


Fig -3.2: What a Computer Sees

Source: medium.com

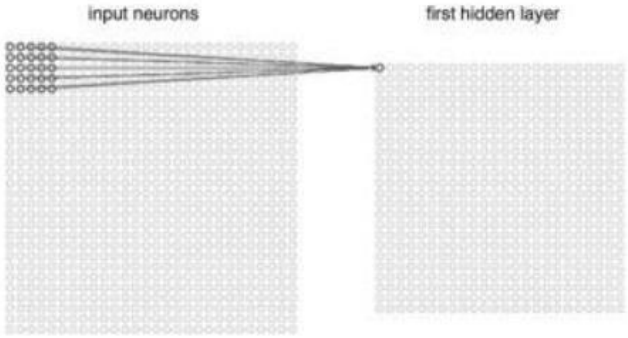
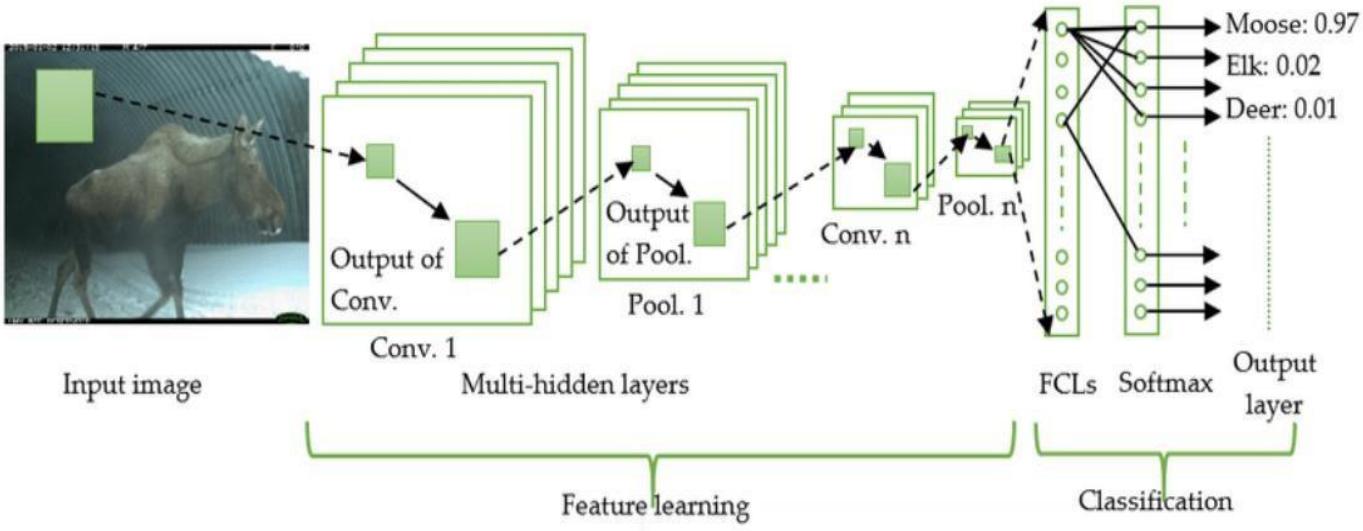
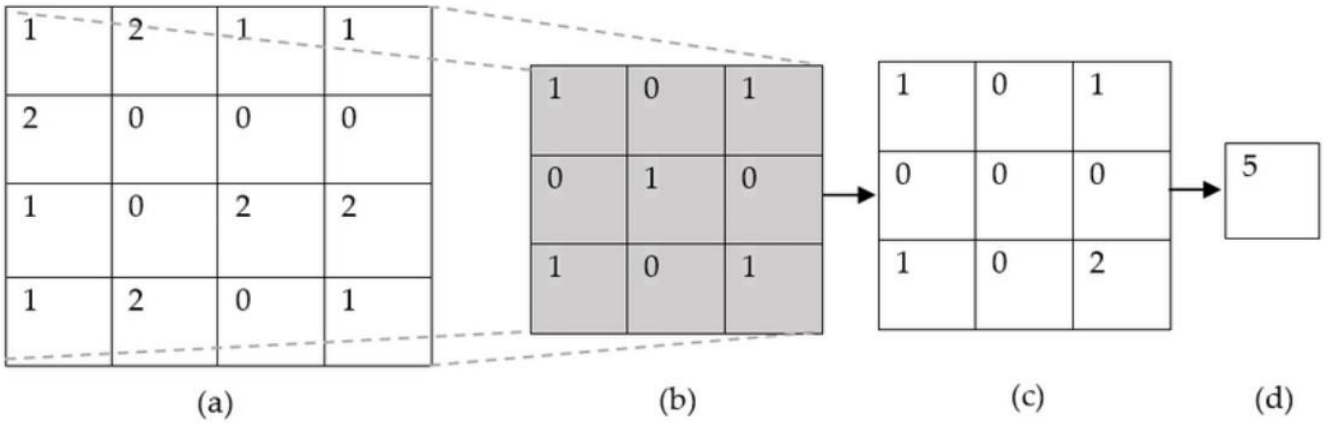


Fig -3.3 : Input neurons and hidden layer

Instead of an image, the computer sees an array of pixels. For example, if image size is 300x300, In this case, the size of the array would be 300x300x3. Where 300 is width, next 300 is height and 3 is RGB values. The computer is assigned a value from 0 to 255 to each of these numbers. This value describes the intensity of the pixel at each point. To solve this problem, the computer looks for the character ristics of the base level. In human understanding such characteristics are seen as, for example the trunk or large ears. For the computer, these characteristics are seen as, boundaries or curvatures. And then through the groups of convolutional layers, computer constructs more abstract concepts. In the above fig, the two images show us what a human eye and computer see when they are presented with an image. the image is passed through a series of convolutional, nonlinear, pooling and fully connected layers, and then the output is generated.

This operation, from a human perspective, is analogous to identifying boundaries and simple colors on the image. But in order to recognize the properties of a higher-level factor such as the trunk or large ears the whole network is needed. The network will consist of several convolutional networks mixed with nonlinear layers and pooling layers. When the image passes through one convolution layer, the output of the first layer becomes the input for the second layer. And this happens with every convolutional layer coming in order. The nonlinear layer is added after each convolution operation. It has an activation function, which brings nonlinear property. Without this property a network will not be sufficiently intense and will not be able to model the response variable. The pooling layer follows the nonlinear layer. It works with width and height of the image and performs a down sampling operation on them. As a result the image volume is reduced. This means that if some features (as for example boundaries) have already been identified in the previous convolution operation, than a detailed image is no longer needed for further processing, and it is compressed to as less detailed pictures.





2.2. EXPERIMENTS AND RESULT

In this work, a CNN model has been developed, to classify different animals, based on a specific business structure deal with Animal classification using a suitable deep learning technique. This model was evaluated by a scientific approach to measure accuracy. Convolutional Neural Network (CNN) is used to build the model. Several experiments have been carried out in order to show the performance of the proposed method and the other state-of the-art methods. In the following subsections, the dataset description and experimental setup and results are presented.

3. MODEL

Import Packages

```
import numpy as np
import os
from sklearn.metrics import confusion_matrix
import seaborn as sn; sn.set(font_scale=1.4)
from sklearn.utils import shuffle
import matplotlib.pyplot as plt
import cv2
import tensorflow as tf
from tqdm import tqdm
```

```
class_names = ['BUFFALO', 'CAT','COW','DEER','DOG', 'ELEPHANT','FOX', 'FROG', 'HORSE', 'LION', 'MONKEY','PIG', 'RABBIT', 'SQUIRRE
class_names_label = {class_name:i for i, class_name in enumerate(class_names)}

nb_classes = len(class_names)

IMAGE_SIZE = (150, 150)
```

Loarding the data

```
def load_data():
    """
     loarding data
    """

    datasets = ['C:/Users/Prabashi/Documents/DL/Class/Train' , 'C:/Users/Prabashi/Documents/DL/Class/Test']
    output = []

    # Iterate through training and test sets
    for dataset in datasets:

        images = []
        labels = []

        print("Loading {}".format(dataset))

        # Iterate through each folder corresponding to a category
        for folder in os.listdir(dataset):
            label = class_names_label[folder]

            # Iterate through each image in our folder
            for file in tqdm(os.listdir(os.path.join(dataset, folder))):

                # Get the path name of the image
                img_path = os.path.join(os.path.join(dataset, folder), file)

                # Open and resize the img
                image = cv2.imread(img_path)
                image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                image = cv2.resize(image, IMAGE_SIZE)

                # Append the image and its corresponding label to the output
```

```
                # Append the image and its corresponding label to the output
                images.append(image)
                labels.append(label)

        images = np.array(images, dtype = 'float32')
        labels = np.array(labels, dtype = 'int32')

        output.append((images, labels))

    return output
```

```
: (train_images, train_labels), (test_images, test_labels) = load_data()
```

```
Loading C:/Users/Prabashi/Documents/DL/Class/Train
100%|| 10/10 [00:00<00:00, 10.04it/s]
100%|| 13/13 [00:00<00:00, 21.13it/s]
100%|| 11/11 [00:00<00:00, 28.83it/s]
100%|| 10/10 [00:00<00:00, 27.43it/s]
100%|| 11/11 [00:00<00:00, 33.30it/s]
100%|| 11/11 [00:00<00:00, 25.89it/s]
100%|| 10/10 [00:00<00:00, 21.78it/s]
100%|| 6/6 [00:00<00:00, 20.64it/s]
100%|| 8/8 [00:00<00:00, 23.28it/s]
100%|| 10/10 [00:00<00:00, 31.10it/s]
100%|| 10/10 [00:00<00:00, 25.59it/s]
100%|| 10/10 [00:00<00:00, 37.10it/s]
100%|| 11/11 [00:00<00:00, 27.57it/s]
100%|| 10/10 [00:00<00:00, 26.09it/s]
100%|| 7/7 [00:00<00:00, 32.57it/s]
Loading C:/Users/Prabashi/Documents/DL/Class/Test
100%|| 4/4 [00:00<00:00, 51.51it/s]
```


3.1. DATA VISUALIZATION.

Let's explore the dataset

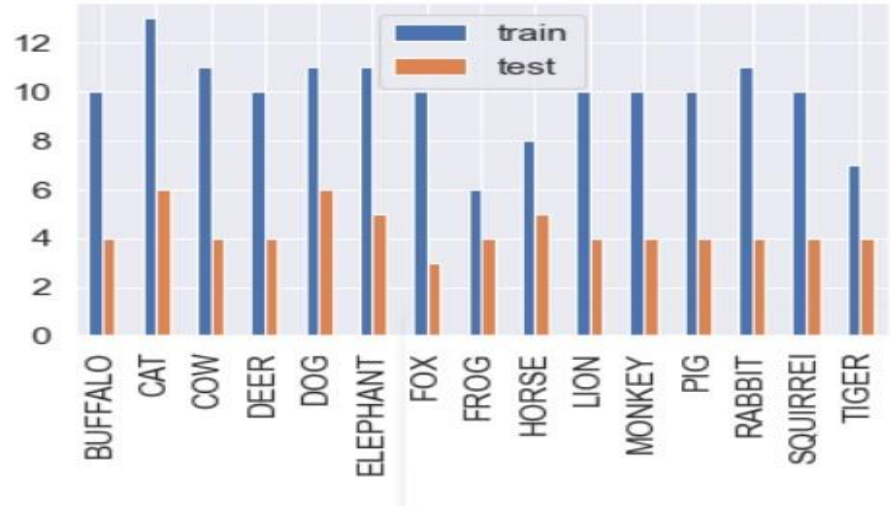
```
: n_train = train_labels.shape[0]
  n_test = test_labels.shape[0]

  print ("Number of training examples: {}".format(n_train))
  print ("Number of testing examples: {}".format(n_test))
  print ("Each image is of size: {}".format(IMAGE_SIZE))
```

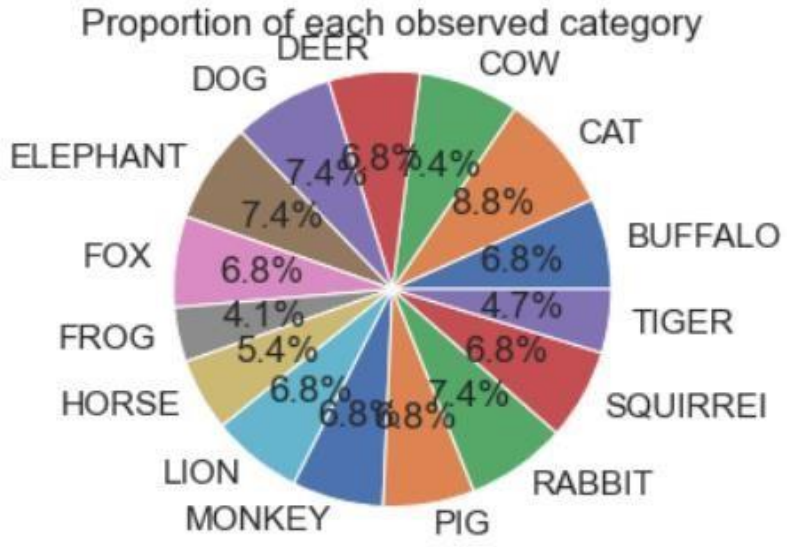
```
Number of training examples: 148
Number of testing examples: 65
Each image is of size: (150, 150)
```

```
import pandas as pd

_, train_counts = np.unique(train_labels, return_counts=True)
_, test_counts = np.unique(test_labels, return_counts=True)
pd.DataFrame({'train': train_counts,
              'test': test_counts},
             index=class_names
            ).plot.bar()
plt.show()
```




```
plt.pie(train_counts,
        explode=(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) ,
        labels=class_names,
        autopct='%1.1f%%')
plt.axis('equal')
plt.title('Proportion of each observed category')
plt.show()
```



```
#Good practice: scale the data

train_images = train_images / 255.0
test_images = test_images / 255.0
```

```
def display_random_image(class_names, images, labels):
    """
        Display a random image from the images array and its correspond label from the labels array.
    """

    index = np.random.randint(images.shape[0])
    plt.figure()
    plt.imshow(images[index])
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.title('Image #{} : '.format(index) + class_names[labels[index]])
    plt.show()
```

```
display_random_image(class_names, train_images, train_labels)
```


```
def display_examples(class_names, images, labels):
    """
        Display 25 images from the images array with its corresponding labels
    """

    fig = plt.figure(figsize=(10,10))
    fig.suptitle("Some examples of images of the dataset", fontsize=16)
    for i in range(25):
        plt.subplot(5,5,i+1)
        plt.xticks([])
        plt.yticks([])
        plt.grid(False)
        plt.imshow(images[i], cmap=plt.cm.binary)
        plt.xlabel(class_names[labels[i]])
    plt.show()
```

```
: display_examples(class_names, train_images, train_labels)
```

Some examples of images of the dataset

3.2. MODEL CONSTRUCTION.

The model is built by using Convolutional neural network. Convolutional neural networks (CNN) are a special architecture of artificial neural networks, proposed by Yann LeCun in 1988. CNN uses some features of the visual cortex. Now that the pre-processing phase is done, neural network can be implemented now. It is going to have 2 convolution layers with 2 x 2 max-pooling. Max-pooling is a technique used to reduce the dimensions of an image by taking the maximum pixel value of a grid. This also helps reduce over fitting and makes the model more generic. Next, add 2 fully connected layers. At the very end of the fully connected layers is a softmax layer.

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu', input_shape = (150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dense(6, activation=tf.nn.softmax)
])
```

Then, we can compile it with some parameters such as:

- Optimizer: adam = RMSProp + Momentum. What is Momentum and RMSProp ?
- Momentum = takes into account past gradient to have a better update.
- RMSProp = exponentially weighted average of the squares of past gradients.
- Loss function: we use sparse categorical crossentropy for classification, each images belongs to one class only

```
model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', metrics=['accuracy'])
```

We fit the model to the data from the training set. The neural network will learn by itself the pattern in order to distinguish each category

```
model.add(tf.keras.layers.Dense(10, activation = 'softmax'))
epochs=10
history = model.fit(train_images, batch_size=12, validation_split = 0.2)
```

```
Train on 11227 samples, validate on 2807 samples
Epoch 1/20
11227/11227 [==============================] - 9s 798us/sample - loss: 1.2029 - acc: 0.5484 - val_loss: 0.9420 - val_acc: 0.646
2
Epoch 2/20
11227/11227 [==============================] - 6s 512us/sample - loss: 0.7826 - acc: 0.7120 - val_loss: 0.8539 - val_acc: 0.677
2
Epoch 3/20
11227/11227 [==============================] - 5s 414us/sample - loss: 0.6219 - acc: 0.7755 - val_loss: 0.7070 - val_acc: 0.747
4
Epoch 4/20
11227/11227 [==============================] - 5s 420us/sample - loss: 0.4614 - acc: 0.8372 - val_loss: 0.7136 - val_acc: 0.747
4
Epoch 5/20
11227/11227 [==============================] - 5s 438us/sample - loss: 0.3462 - acc: 0.8829 - val_loss: 0.7459 - val_acc: 0.741
4
Epoch 6/20
11227/11227 [==============================] - 5s 436us/sample - loss: 0.2944 - acc: 0.9009 - val_loss: 0.6843 - val_acc: 0.781
3
Epoch 7/20
11227/11227 [==============================] - 5s 426us/sample - loss: 0.1656 - acc: 0.9498 - val_loss: 0.7731 - val_acc: 0.762
0
Epoch 8/20
11227/11227 [==============================] - 5s 419us/sample - loss: 0.1188 - acc: 0.9670 - val_loss: 0.7210 - val_acc: 0.789
1
Epoch 9/20
11227/11227 [==============================] - 5s 423us/sample - loss: 0.0765 - acc: 0.9817 - val_loss: 0.8328 - val_acc: 0.791
2
Epoch 10/20
```

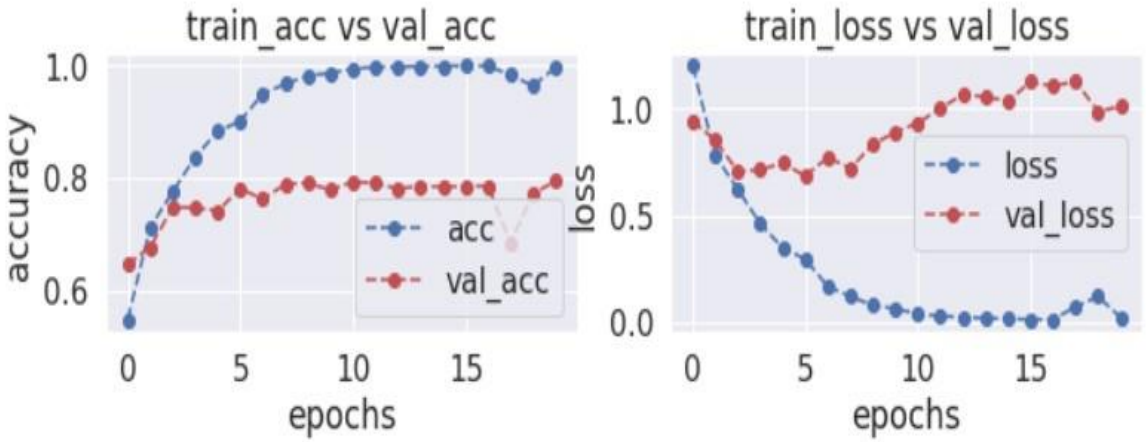
```
def plot_accuracy_loss(history):
    """
        Plot the accuracy and the loss during the training of the nn.
    """
    fig = plt.figure(figsize=(10,5))

    # Plot accuracy
    plt.subplot(221)
    plt.plot(history.history['acc'],'bo--', label = "acc")
    plt.plot(history.history['val_acc'], 'ro--', label = "val_acc")
    plt.title("train_acc vs val_acc")
    plt.ylabel("accuracy")
    plt.xlabel("epochs")
    plt.legend()

    # Plot loss function
    plt.subplot(222)
    plt.plot(history.history['loss'],'bo--', label = "loss")
    plt.plot(history.history['val_loss'], 'ro--', label = "val_loss")
    plt.title("train_loss vs val_loss")
    plt.ylabel("loss")
    plt.xlabel("epochs")

    plt.legend()
    plt.show()
```

```
: plot_accuracy_loss(history)
```



3.3. MODEL TESTING AND EVALUATION.

Once the model has been trained, model testing can be carried out. During this phase a test set of data is loaded. This data set has never been seen by the model and therefore its true accuracy will be verified. Finally, the saved model is ready to be used in the real world. The name of this phase is called model evaluation. This means that the model can be used to evaluate new data. In the below fig, a tiny code has been used to show that the testing is done appropriately using the images in the desktop.

Let's see how the classifier is doing on random images.

```
: predictions = model.predict(test_images)     # Vector of probabilities
  pred_labels = np.argmax(predictions, axis = 1) # We take the highest probability

  display_random_image(class_names, test_images, pred_labels)
```

```
3/3 [==============================] - 0s 49ms/step
```

4. CONCLUSION AND FUTURE WORK

This system comes under deep learning which is advanced technique at present. CNN is more suitable for image processing especially in image classification. It concludes the experimental result what which is obtained from developed system is comparatively more accurate than many procedures followed before. Image recognition and prediction is a wide concept which is very deep. Deep learning concepts can be used in this subject to get efficient results. Concept like Convolutional neural networks(CNN) can be used, which is effective in obtaining results for image identification and classification