MINI PROJECT
(2020-21)

ON

Image Classification

MID-TERM REPORT

Institute of Engineering & Technology

Submitted by-
Saurabh Agarwal
(181500635)
Akash Sharma
(181500059)
Shivansh Agrawal
(181500681)
Raj Gupta
(181500542)

Supervised By: -
Mr. Sharad Gupta sir
Department of Computer Engineering & Application

Contents

Acknowledgment

In the present world of competition there is a race of existence in which those are having will to come forward succeed. Project is like a bridge between theoretical and practical working. With this willing we joined this particular project. It gives us great pleasure and we are glad to represent the synopsis of our B. Tech mini Project undertaken during the third year of our graduation era.

It has indeed been a great privilege for us to have Mr. Sharad Gupta Department of Computer Engineering & Application, Institute of Engineering & Technology, GLA University, Mathura, as our mentor in this Project. His awe-inspiring personality, superb guidance and constant encouragement are the motive force behind this project. We take this opportunity to express our utmost gratitude to him. We are also indebted to him for his timely and valuable advise.

We are highly grateful to Prof. Anand Singh Jalal, Head, Department of Computer Engineering & Application for providing necessary facilities and encouraging us during the course of work. We are thankful to all technical and non-technical staff of the Department of the Computer Science and Engineering for their constant assistance and their co-operation.

Abstract

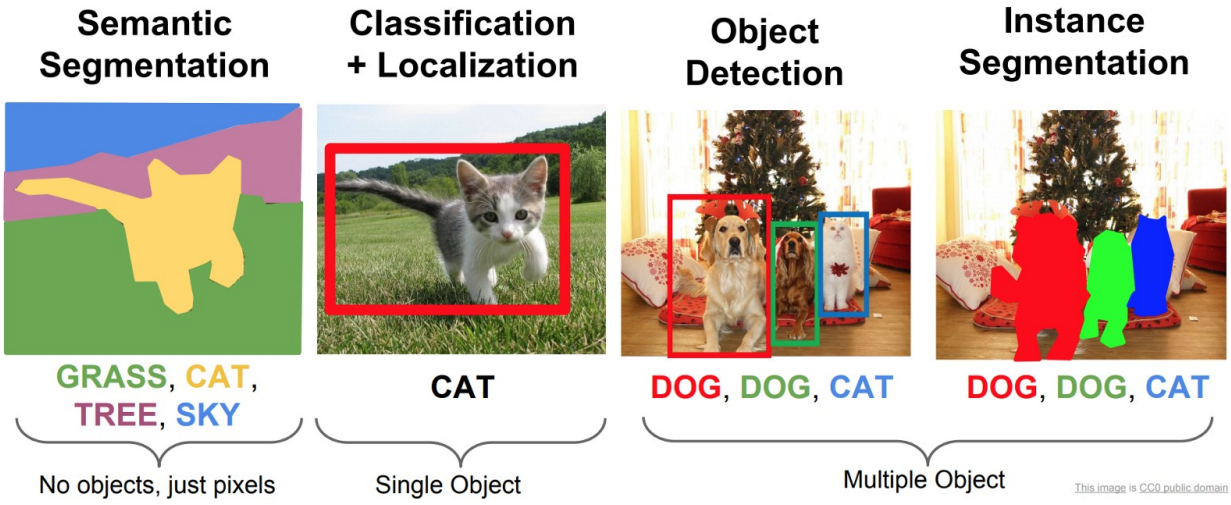
Image classification is a fundamental problem in computer vision. Deep learning provides successful results for machine learning problems. Many algorithms like minimum distance algorithm, K-Nearest neighbour algorithm, Nearest Clustering algorithm, Fuzzy C-Means algorithm, Maximum likelihood algorithm are used for the purpose of image classification. In this paper, image classification is performed using convolutional neural network. Generally convolutional neural network uses GPU technology because of huge number of computations but, in proposed method we are building a very small network which can work on CPU as well. The network is trained using a subset of Kaggle Dog-Cat dataset. This trained classifier can classify the given image into either cat or dog. The same network can train with any other dataset and classify the images into one of the two predefined class.

Convolutional Neural Networks have become state of the art methods for image classification over the last couple of years. By now they perform better than human subjects on many of the image classification datasets. Most of these datasets are based on the notion of concrete classes (i.e. images are classified by the type of object in the image). The classification performance of popular CNN architectures is evaluated on this dataset and variations of the dataset that might be interesting for further research are identified.

Introduction

Classification is a systematic arrangement in groups and categories based on its features. Image classification came into existence for decreasing the gap between the computer vision and human vision by training the computer with the data. The image classification is achieved by differentiating the image into the prescribed category based on the content of the vision. The conventional methods used for image classifying is part and piece of the field of artificial intelligence (AI) formally called as machine learning. The machine learning consists of feature extraction module that extracts the important features such as edges, textures, etc. and a classification module that classify based on the features extracted. The main limitation of machine learning is, while separating, it can only extract certain set of features on images and unable to extract differentiating features from the training set of data. This disadvantage is rectified by using the deep learning. Deep learning (DL) is a sub field to the machine learning, capable of learning through its own method of computing. A deep learning model is introduced to persistently break down information with a homogeneous structure like how a human would make determinations. To accomplish this, deep learning utilizes a layered structure of several algorithms expressed as an artificial neural system (ANN). The architecture of an ANN is simulated with the help of the biological neural network of the human brain. This makes the deep learning most capable than the standard machine learning models. In deep learning, we consider the neural networks that identify the image based on its features. This is accomplished for the building of a complete feature extraction model which is capable of solving the difficulties faced due to the conventional methods. The extractor of the integrated model should be able to learn extracting the differentiating features from the training set of images accurately.

The process of image classification involves two steps, training of the system followed by testing. The training process means, to take the characteristic properties of the images (form a class) and form a unique description for a particular class. The process is done for all classes depending on the type of classification problem; binary classification or multi-class classification. The testing step means to categorize the test images under various classes for which system was trained. This assigning of class is done based on the partitioning between classes based on the training features.



Project Description

Convolutional neural networks (CNN) is a special architecture of artificial neural networks. CNN uses some features of the visual cortex. One of the most popular uses of this architecture is image classification. The main task of image classification is acceptance of the input image and the following definition of its class. This is a skill that people learn from their birth and are able to easily determine that the image in the picture is an elephant. But the computer sees the pictures quite differently:

What I see

What a computer sees

Instead of the image, the computer sees an array of pixels. For example, if image size is 300×300. In this case, the size of the array will be 300×300×3. Where 300 is width, next 300 is height and 3 is RGB channel 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.

The image is passed through a series of convolutional, nonlinear, pooling layers and fully connected layers, and then generates the output. The Convolution layer is always the first. The image (matrix with pixel values) is entered into it. Imagine that the reading of the input matrix begins at the top left of the image. Next the software selects a smaller matrix there, which is called a filter (or neuron, or core). Then the filter produces convolution, i.e. moves along the input image. The filter's task is to multiply its values by the original pixel values. All these multiplications are summed up. One number is obtained in the end. Since the filter has read the image only in the upper left corner, it moves further and further right by 1 unit performing a similar operation. After passing the filter across all positions, a matrix is obtained, but smaller than an input matrix.

The network will consist of several convolutional networks mixed with nonlinear 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 further convolutional layer.

The nonlinear layer is added after each convolution operation. It has an activation function, which brings nonlinear property. Without this property a network would not be sufficiently intense and will not be able to model the response variable (as a class label).

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, then a detailed image is no longer needed for further processing, and it is compressed to less detailed pictures.

After completion of a series of convolutional, nonlinear and pooling layers, it is necessary to attach a fully connected layer. This layer takes the output information from convolutional networks. Attaching a fully connected layer to the end of the network results in an N dimensional vector, where N is the amount of classes from which the model selects the desired class. A fragment of the code of this model written in Python will be considered further in the practical part.

Hardware Requirements

- Processor: Intel dual core or above
- Processor Speed: 1.0GHZ or above
- RAM: 1 GB RAM or above
- Hard Disk: 20 GB hard disk or above

Software Requirements

- Language: Python
- Framework: Keras
- Libraries: Numpy, MatplotLib, Pandas
- Dataset: Celebrity images (41 categories)

Process

To create such model, it is necessary to go through the following phases:

- Model construction
- Model training
- Model testing
- Model evaluation

Model construction depends on machine learning algorithms. In this project case, it was neural networks. Such an algorithm looks like:

- begin with its object: model = Sequential()
- Then consist of layers with their types: model.add (*type_of_layer ()*)
- After adding a sufficient number of layers the model is compiled. At this
- Moment Keras communicates with TensorFlow for construction of the model. During model compilation it is important to write a loss function and an optimizer algorithm.
- It looks like: model.compile (loss= 'name_of_loss_function', optimizer= 'name_of_opimazer_alg') the loss function shows the accuracy of each prediction made by the model.

Before model training it is important to scale data for their further use. After model construction it is time for model training. In this phase, the model is trained using training data and expected output for this data.

It looks this way: model.fit (training_data, expected_output).

Progress is visible on the console when the script runs. At the end it will report the final accuracy of the model. Once the model has been trained it is possible to carry out model testing. During this phase a second set of data is loaded. This data set has never been seen by the model and therefore its true accuracy will be verified. After the model training is complete, and it is understood that the model shows the right result.

Finally, the saved model can be used in the real world. The name of this phase is model evaluation. This means that the model can be used to evaluate new data.

Comparison Between Different Algorithms :-

- ## Logistic Regression:-

Simple logistic regression is a statistical method that can be used for binary classification problems. In the context of image processing, this could mean identifying whether a given image belongs to a particular class (y=1) or not (y=0), e.g. "cat" or “dog". A logistic regression algorithm takes as its input a feature vector x and outputs a probability, y^ = P(y=1|x) that the feature vector represents an object belonging to the class. For images, the feature vector might be just the values of the red, green and blue (RGB) channels for each pixel in the image: a one-dimensional array of nx = nheight × nwidth × 3 real numbers formed by flattening the three-dimensional array of pixel RGB values. A logistic regression model is so named because it calculates y^=σ(z)

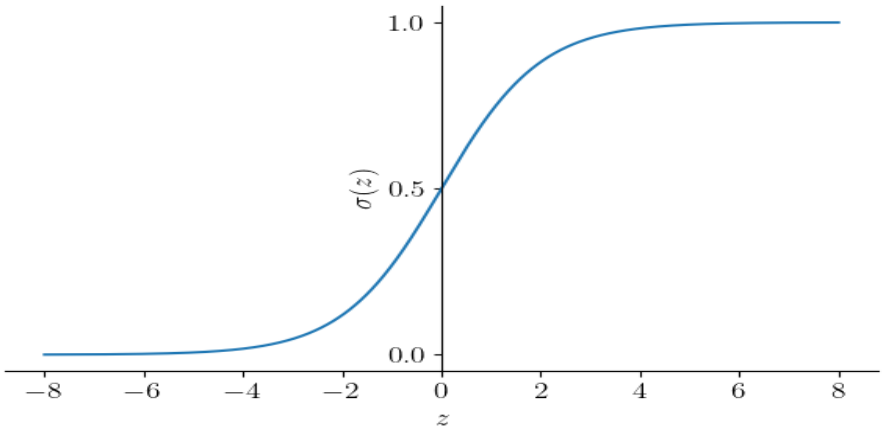
where

$$\sigma(z)=1/1+e-z$$

is the logistic function and

$$z=wTx+b$$

for a set of parameters, w and b. w is a nx-dimensional vector (one component for each component of the feature vector) and b is a constant "bias".



Training a logistic regression algorithm involves obtaining the optimum values of wand b such that y^(i) most closely predicts y(i) for a set of m provided, pre-classified examples (i.e. m images corresponding to feature vectors x(i) for which the classification y(i) is known): this is a supervised learning technique. In practice, this usually means calculating the l*oss function*,

$$L(y^{(i)}, y(i))) = -[y(i) \log y^{(i)} + (1-y(i)) \log(1-y^{(i)})]$$

for each training example, i, and minimizing the *cost function*,

$$J(w, b) = (1/m) \sum_{i=1 \text{ to } m} (L(y^{(i)}, y(i))))$$

across all m training examples. The loss function captures, in a way suitable for numerical minimization of J, the difference between the predicted and actual classification of each training example. It can be shown that

$$\partial L/\partial wj = (y^{(i)} - y(i)) * x(i)j \text{ and } \partial L/\partial b = y^{(i)}-y(i) ,$$

where j=1,2,…,nx labels the components of the feature vector.

In numerically minimizing J(w, b)one starts with an initial guess for wj and b and uses these expressions to determine how to change them iteratively so that J keeps decreasing. That is, on each iteration the values of the parameters are changed according to descent along the steepest gradient:

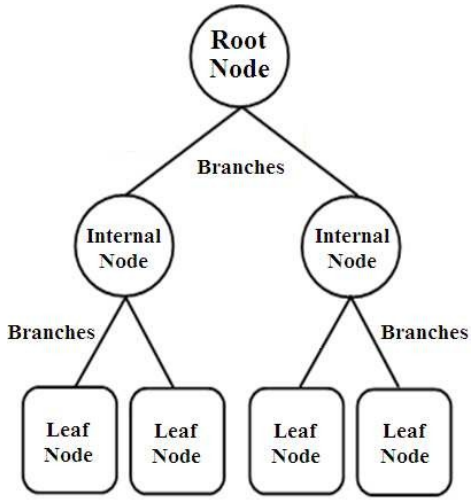
$$wj \rightarrow wj - \alpha (\partial J/\partial wj) = wj- (\alpha/m) \sum_{i=1 \text{ to } m} (\partial L/\partial wj)$$

and similarly for b, where α is some learning rate that determines how large each step taken in the direction of greatest decrease in J is. Choosing a suitable value for α is a subtle art (too small and the training is slow, too large and the steps taken in gradient descent are too large and the training may not converge reliably on the minimum in J), but for small, simple problems can be determined by trial-and-error.

- **Decision Tree :-**

The decision tree consists of basic elements, which:

- Root
- Sub nodes
- Leaves



(a)

The construction of the decision tree starts from the root and then to the sub-nodes. The nodes represent distinct features, which are the decision points, on the basis of it, the information is classified. The nodes at different levels are linked to each other by branches,

which represent different decisions taken by testing the status of the feature in the node. Thus, during the classification process, the path from the root to the nodes is followed to test the feature and make decisions along the path. This path must comply with one distinct rule/feature. In the image classification process, the decision tree is created by repeated division of spectral distribution of the training data set which entered for this classification process. During the classification process, representative spectral values are determined and mapping to a large number of classification categories. The decision tree aims to divide the data set into a homogeneous group as possible in terms of the expected variable, that meaning each end of a node (leaf) is a decision (category or class)

Building a decision tree requires significant training and a training data set. The training data consists of two types of variables.

- Response variables
- Explanatory variables

The decision tree is based on several algorithms that together formed the mathematical basis from which the decision tree can be built. These algorithms are the basis for most of the algorithms in which the decision tree operates, and through which the decision tree can be built and made the required classifications.

Basic Algorithms

The decision tree is based on several algorithms that together formed the mathematical basis from which the decision tree can be built. These algorithms are the basis for most of the algorithms in which the decision tree operates, and through which the decision tree can be built and made the required classifications.

Entropy

It is considered one of the most applied algorithms, as it determines the amount of information that the event gives, the lower the possibility of the event (rarely) the more information it provides. In general, the algorithm can be simplified as follows

If the probability distribution is given P as P = (p1, p2, P3, ……, Pn),

And give a sample S,

The information carried by this distribution is called the Entropy of P and is given by the following equation,

$$Entropie\ (P) = -\sum i=1 \text{ to } n\ (Pi \times \log\ (Pi))$$

Where (Pi) is the probability that the number (i) will appear during the flow/analysis process.

Gain Information

This algorithm allows measuring the degree of mixing of the classes for all samples in classification as well as anywhere in the tree structure. It specifies a function to select the test that is supposed to label the current node in the tree.

It defines the amount of gain for a test (T) and a position (P), By the following equation,

$$Gain(P, T) = Entropie(P) - \sum j=1 \text{ to } n\ (Pj \times Entroie(Pj))$$

Where (Pj) is a set of all possible values of the (T) attribute.

These equations can be used to rank attributes and build a decision tree, as each node takes an attribute with the highest information gain value among attributes that not yet considered in the path (from the root to the nodes).

The decision tree is one of the effective and practical classification techniques in the field of remote sensing images classification, and it has been used in various projects, such as urban change mapping, land use maps, land surface study maps and extracting information from remote sensing images, This technique has shown very high accuracy in classification processes that distinguish it from other technologies. It is also characterized by flexibility and ability to deal with non-linear relationships between features and categories, and this, in turn, leads to a greater degree of classification accuracy. In the field of remote sensing, due to the complex distribution of data features or the presence of a statistical distribution of data dimensions and different scales, decision tree classification can obtain the best results due to its flexibility, and the durability of the data distribution feature and relationships classification.

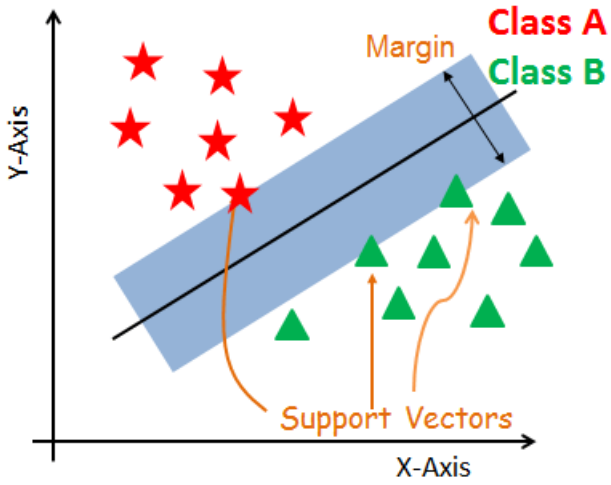
A disadvantage of the decision tree is that it is unstable when changing the area of features and training areas, as well as sometimes it can be complex and requires a large amount of computing operations.

- **Support Vector Machine :-**

SVM offers very high accuracy compared to other classifiers such as logistic regression, and decision trees. It is known for its kernel trick to handle nonlinear input spaces. It is used in a variety of applications such as face detection, intrusion detection, classification of emails, news articles and web pages, classification of genes, and handwriting recognition.

SVM is an exciting algorithm and the concepts are relatively simple. The classifier separates data points using a hyperplane with the largest amount of margin. That's why an SVM classifier is also known as a discriminative classifier. SVM finds an optimal hyperplane which helps in classifying new data points.

Generally, Support Vector Machines is considered to be a classification approach, it but can be employed in both types of classification and regression problems. It can easily handle multiple continuous and categorical variables. SVM constructs a hyperplane in multidimensional space to separate different classes. SVM generates optimal hyperplane in an iterative manner, which is used to minimize an error. The core idea of SVM is to find a maximum marginal hyperplane(MMH) that best divides the dataset into classes.



Support Vectors

Support vectors are the data points, which are closest to the hyperplane. These points will define the separating line better by calculating margins. These points are more relevant to the construction of the classifier.

Hyperplane

A hyperplane is a decision plane which separates between a set of objects having different class memberships.

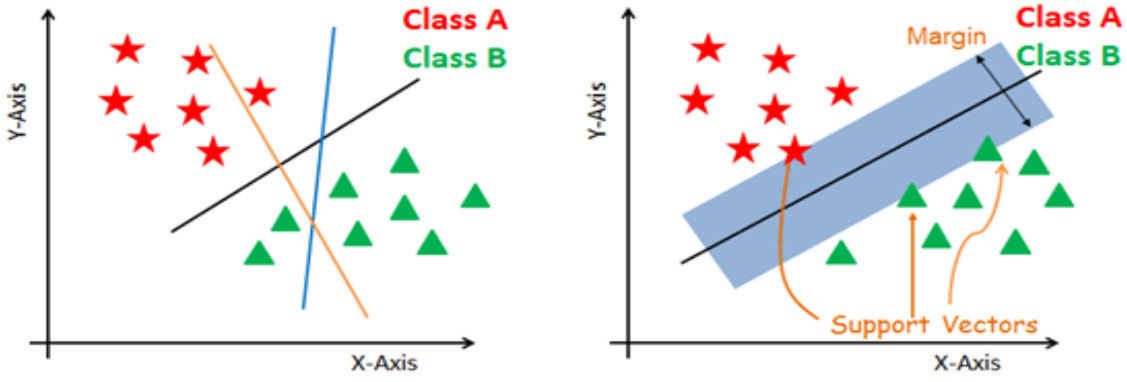
Margin

A margin is a gap between the two lines on the closest class points. This is calculated as the perpendicular distance from the line to support vectors or closest points. If the margin is larger in between the classes, then it is considered a good margin, a smaller margin is a bad margin.

How does SVM work?

The main objective is to segregate the given dataset in the best possible way. The distance between the either nearest points is known as the margin. The objective is to select a hyperplane with the maximum possible margin between support vectors in the given dataset. SVM searches for the maximum marginal hyperplane in the following steps

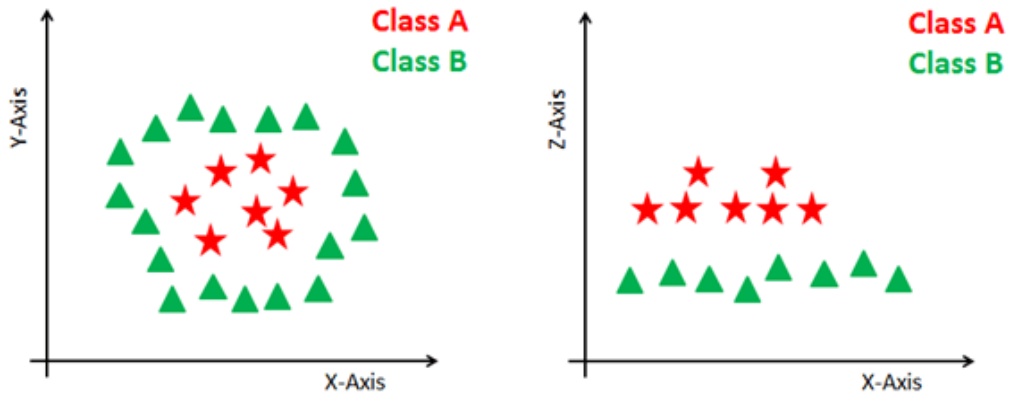
- Generate hyperplanes which segregates the classes in the best way. Left-hand side figure showing three hyperplanes black, blue and orange. Here, the blue and orange have higher classification error, but the black is separating the two classes correctly.
- Select the right hyperplane with the maximum segregation from the either nearest data points as shown in the right-hand side figure.



Dealing with non-linear and inseparable planes

Some problems can't be solved using linear hyperplane, as shown in the figure below (left-hand side).

In such situation, SVM uses a kernel trick to transform the input space to a higher dimensional space as shown on the right. The data points are plotted on the x-axis and z-axis (Z is the squared sum of both x and y: z=x^2=y^2). Now you can easily segregate these points using linear separation.



SVM Kernels

The SVM algorithm is implemented in practice using a kernel. A kernel transforms an input data space into the required form. SVM uses a technique called the kernel trick. Here, the kernel takes a low-dimensional input space and transforms it into a higher dimensional space. In other words, you can say that it converts non-separable problem to separable problems by adding more dimension to it. It is most useful in non-linear separation problem. Kernel trick helps you to build a more accurate classifier.

- **Convolutional Neural Network(CNN) :-**

The convolutional neural network (CNN) is a class of deep learning neural network. CNNs represent a huge breakthrough in image recognition. They're most commonly used to analyze visual imagery and are frequently working behind the scenes in image classification. They can be found at the core of everything from Facebook's photo tagging to self-driving cars. They're working hard behind the scenes in everything from healthcare to security.

They're fast and they're efficient. But how do they work?

Image classification is the process of taking an input (like a picture) and outputting a class (like "cat") or a probability that the input is a particular class ("there's a 90% probability that this input is a cat"). You can look at a picture and know that you're looking at a terrible shot of your own face, but how can a computer learn to do that?

A CNN has

- convolutional layers
- ReLU layers
- pooling layers
- a fully connected layers

A classic CNN architecture would look something like this:

Input ->Convolution ->ReLU ->Convolution ->ReLU ->Pooling ->
ReLU ->Convolution ->ReLU ->Pooling ->Fully Connected

A CNN convolves learned features with input data and uses 2D convolutional layers. This means that this type of network is ideal for processing 2D images. Compared to other image classification algorithms, CNNs actually use very little pre-processing. This means that they can learn the filters that have to be hand-made in other algorithms. CNNs can be used in tons of applications from image and video recognition, image classification, and recommender systems to natural language processing and medical image analysis.

CNNs have an input layer, and output layer, and hidden layers. The hidden layers usually consist of convolutional layers, ReLU layers, pooling layers, and fully connected layers.

- Convolutional layers apply a convolution operation to the input. This passes the information on to the next layer.
- Pooling combines the outputs of clusters of neurons into a single neuron in the next layer.

- Fully connected layers connect every neuron in one layer to every neuron in the next layer.

In a convolutional layer, neurons only receive input from a subarea of the previous layer. In a fully connected layer, each neuron receives input from *every* element of the previous layer.

A CNN works by extracting features from images. This eliminates the need for manual feature extraction. The features are not trained! They're learned while the network trains on a set of images. This makes deep learning models extremely accurate for computer vision tasks. CNNs learn feature detection through tens or hundreds of hidden layers. Each layer increases the complexity of the learned features.

A CNN

- starts with an input image
- applies many different filters to it to create a feature map
- applies a ReLU function to increase non-linearity
- applies a pooling layer to each feature map
- flattens the pooled images into one long vector.
- inputs the vector into a fully connected artificial neural network.
- Processes the features through the network. The final fully connected layer provides the "voting" of the classes that we're after.
- Trains through forward propagation and back-propagation for many, many epochs. This repeats until we have a well-defined neural network with trained weights and feature detectors.

So what does that mean?

At the very beginning of this process, an input image is broken down into pixels.

For a black and white image, those pixels are interpreted as a 2D array (for example, 2x2 pixels). Every pixel has a value between 0 and 255. (Zero is completely black and 255 is completely white. The greyscale exists between those numbers.) Based on that information, the computer can begin to work on the data.

For a color image, this is a 3D array with a blue layer, a green layer, and a red layer. Each one of those colors has its own value between 0 and 255. The color can be found by combining the values in each of the three layers.

What are the basic building blocks of a CNN?

Convolutional

The main purpose of the convolution step is to extract features from the input image. The convolutional layer is always the first step in a CNN.

You have an input image, a feature detector, and a feature map. You take the filter and apply it pixel block by pixel block to the input image. You do this through the multiplication of the matrices.

Let's say you have a flashlight and a sheet of bubble wrap. Your flashlight shines a 5-bubble x 5-bubble area. To look at the entire sheet, you would slide your flashlight across each 5x5 square until you'd seen all the bubbles.

The light from the flashlight here is your filter and the region you're sliding over is the receptive field. The light sliding across the receptive fields is your flashlight convolving. Your filter is an array of numbers (also called weights or parameters). The distance the light from your flashlight slides as it travels (are you moving your filter over one row of bubbles at a time? Two?) is called the stride. For example, a stride of one means that you're moving your filter over one pixel at a time. The convention is a stride of two.

The depth of the filter has to be the same as the depth of the input, so if we were looking at a color image, the depth would be 3. That makes the dimensions of this filter 5×5x3. In each position, the filter multiplies the values in the filter with the original values in the pixel. This is element wise multiplication. The multiplications are summed up, creating a single number. If you started at the top left corner of your bubble wrap, this number is representative of the top left corner. Now you move your filter to the next position and repeat the process all around the bubble wrap.

You'll specify parameters like the number of filters, the filter size, the architecture of the network, and so on. The CNN learns the values of the filters on its own during the training process. You have a lot of options that you can work with to make the best image classifier possible for your task. You can choose to pad the input matrix with zeros (zero padding) to apply the filter to bordering elements of the input image matrix. This also allows you to control the size of the feature maps. Adding zero padding is wide convolution. Not adding zero padding is narrow convolution.

This is basically how we detect images! We don't look at every single pixel of an image. We see features like a hat, a red dress, a tattoo, and so on. There's so much information going into our eyes at all times that we couldn't possibly deal with every single pixel of it. We're allowing our model to do the same thing.

The result of this is the convolved feature map. It's smaller than the original input image. This makes it easier and faster to deal with. Do we lose information? Some, yes. But at the same time, the purpose of the feature detector is to detect features, which is exactly what this does.

We create many feature maps to get our first convolutional layer. This allows us to identify many different features that the program can use to learn.

Feature detectors can be set up with different values to get different results. For example, a filter can be applied that can sharpen and focus an image or blur an image. That would give equal importance to all the values. You can do edge enhancement, edge detection, and more. You would do that by applying different feature detectors to create different feature maps. The computer is able to determine which filters make the most sense and apply them.

The primary purpose here is to find features in your image, put them into a feature map, and still preserve the spatial relationship between pixels. That's important so that the pixels don't get all jumbled up.

ReLU layer

The ReLU (rectified linear unit) layer is another step to our convolution layer. You're applying an activation function onto your feature maps to increase non-linearity in the network. This is because images themselves are highly non-linear! It removes negative values from an activation map by setting them to zero.

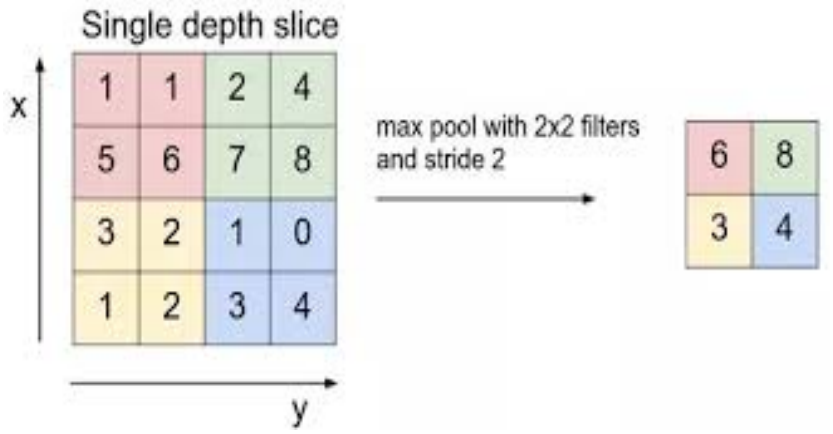
Convolution is a linear operation with things like element wise matrix multiplication and addition. The real-world data we want our CNN to learn will be non-linear. We can account for that with an operation like ReLU. You can use other operations like tanh or sigmoid. ReLU, however, is a popular choice because it can train the network faster without any major penalty to generalization accuracy.

Pooling

The last thing you want is for your network to look for one specific feature in an exact shade in an exact location. That's useless for a good CNN! You want images that are flipped, rotated, squashed, and so on. You want lots of pictures of the same thing so that your network can recognize an object (say, a leopard) in all the images. No matter what the size or location. No matter what the lighting or the number of spots, or whether that leopard is fast asleep or crushing prey. You want **spatial variance**! You want flexibility. That's what pooling is all about.

Pooling progressively reduces the size of the input representation. It makes it possible to detect objects in an image no matter where they're located. Pooling helps to reduce the number of required parameters and the amount of computation required. It also helps control **overfitting**.

Without variance, your network will be useless with images that don't exactly match the training data. **Always, always, always keep your training and testing data separate**! If you test with the data you trained on, your network has the information memorized! It will do a terrible job when it's introduced to any new data.



Overfitting is not cool.

So for this step, you take the **feature map**, apply a **pooling layer**, and the result is the **pooled feature map**.

The most common example of pooling is **max pooling**. In max pooling, the input image is partitioned into a set of areas that don't overlap. The outputs of each area are the maximum value in each area. This makes a smaller size with fewer parameters.

Max pooling is all about grabbing the maximum value at each spot in the image. This gets rid of 75% of the information that is not the feature. By taking the maximum value of the pixels, you're accounting for distortion. If the feature rotates a little to the left or right or whatever, the pooled feature will be the same. You're reducing the size and parameters. This is great because it means that the model won't overfit on that information.

You could use **average pooling** or **sum pooling**, but they aren't common choices. Max pooling tends to perform better than both in practice. In max pooling, you're taking the largest pixel value. In average pooling, you take the average of all the pixel values at that spot in the image.

Flattening

This is a pretty simple step. You flatten the pooled feature map into a sequential column of numbers (a long vector). This allows that information to become the input layer of an artificial neural network for further processing.

Fully connected layer

At this step, we add an artificial neural network to our convolutional neural network.

The main purpose of the artificial neural network is to combine our features into more attributes. These will predict the classes with greater accuracy. This combines features and attributes that can predict classes better.

At this step, the error is calculated and then back-propagated. The weights and feature detectors are adjusted to help optimize the performance of the model. Then the process happens again and again and again. This is how our network trains on the data!

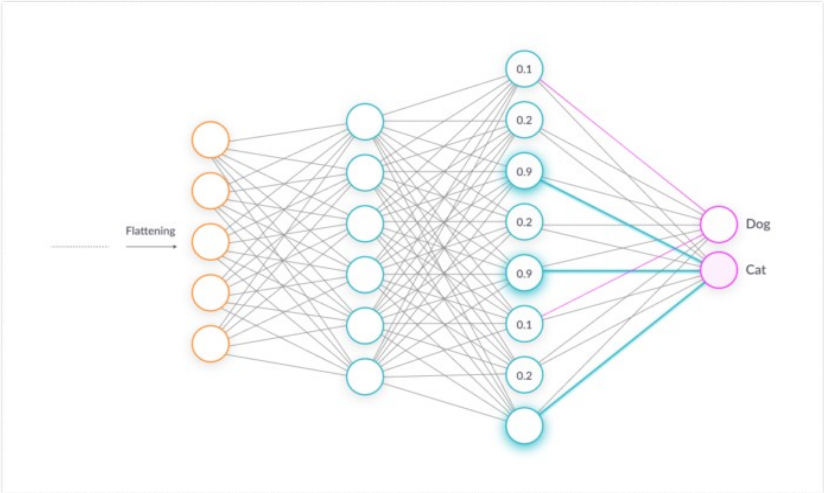
How do the output neurons work when there's more than one?

First, we have to understand what weights to apply to the synapses that connect to the output. We want to know which of the previous neurons are important for the output.

If, for example, you have two output classes, one for a cat and one for a dog, a neuron that reads "0" is absolutely uncertain that the feature belongs to a cat. A neuron that reads "1 is absolutely certain that the feature belongs to a cat. In the final fully connected layer, the

neurons will read values between 0 and 1. This signifies different levels of certainty. A value of 0.9 would signify a certainty of 90%.

The cat neurons that are certain when a feature is identified know that the image is a cat. They say the mathematical equivalent of, “These are my neurons! I should be triggered!” If this happens many times, the network learns that when certain features fire up, the image is a cat.



Once the network has been trained, you can pass in an image and the neural network will be able to determine the image class probability for that image with a great deal of certainty.

The fully connected layer is a traditional Multi-Layer Perceptron. It uses a classifier in the output layer. The classifier is usually a softmax activation function. Fully connected means every neuron in the previous layer connects to every neuron in the next layer. What’s the purpose of this layer? To use the features from the output of the previous layer to classify the input image based on the training data.

Once your network is up and running you can see, for example, that you have a 95% probability that your image is a dog and a 5% probability that your image is a cat.

There isn't anything that says that these two outputs are connected to each other. What is it that makes them relate to each other? Essentially, they wouldn't, but they do when we introduce **the softmax function**. This brings the values between 0 and 1 and makes them add up to 1 (100%). The softmax function takes a vector of scores and squashes it to a vector of values between 0 and 1 that add up to 1.

After you apply a softmax function, you can apply the loss function. Cross entropy often goes hand in hand with softmax. We want to minimize the loss function so we can maximize the performance of our network.

At the beginning of back-propagation, your output values would be tiny. That's why you might choose cross entropy loss. The gradient would be very low and it would be hard for the neural network to start adjusting in the right direction. Using cross entropy helps the network assess even a tiny error and get to the optimal state faster.

Algorithm	Overall Accuracy
K-Nearest Neighbors (k=3)	82.90%
Support Vector Machine	84.68%
Logistic Regression	85.39%
Decision Tree	70.53%
CNN Model	88.5%

Coding and Screenshots

Importing the libraries:-

- import numpy as np
- import pandas as pd
- from keras.preprocessing.image import ImageDataGenerator, load_img
- from keras.utils import to_categorical
- from sklearn.model_selection import train_test_split
- import matplotlib.pyplot as plt
- import random
- import image
- import os
- import PIL

Define Constants:-

- FAST_RUN = False
- IMAGE_WIDTH=128
- IMAGE_HEIGHT=128
- IMAGE_SIZE=(IMAGE_WIDTH, IMAGE_HEIGHT)

- IMAGE_CHANNELS=3

Prepare Training Data:-

- filenames = os.listdir("C:/Users/sa251/Downloads/dogs-vs-cats/train")
- categories = []
- for filename in filenames:
    category = filename.split('.')[0]
    if category == 'dog':
        categories.append(1)
    else:
        categories.append(0)
- df = pd. DataFrame({'filename': filenames,'category': categories})
- df.head()

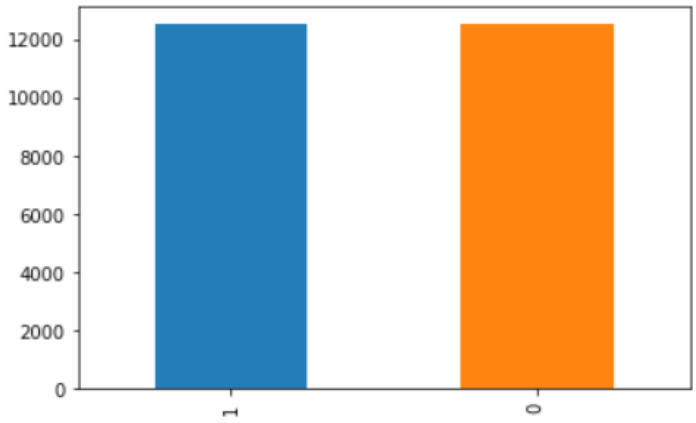
	filename	category
0	cat.0.jpg	0
1	cat.1.jpg	0
2	cat.10.jpg	0
3	cat.100.jpg	0
4	cat.1000.jpg	0

- df.tail()

	filename	category
24995	dog.9995.jpg	1
24996	dog.9996.jpg	1
24997	dog.9997.jpg	1
24998	dog.9998.jpg	1
24999	dog.9999.jpg	1

See Total In count:-

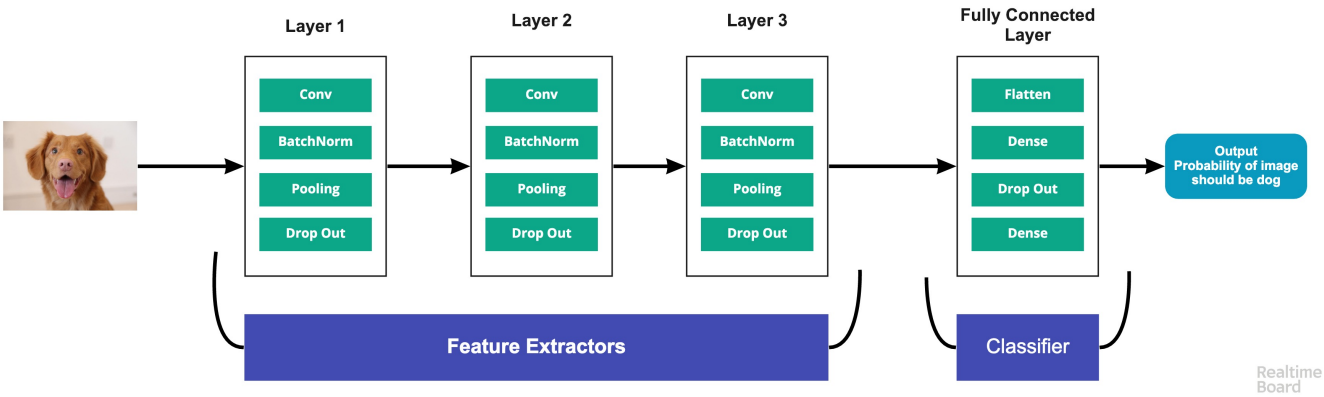
- df['category'].value_counts().plot.bar()



See sample image:-

- sample = random.choice(filenames)
- image = load_img("C:/Users/sa251/Downloads/dogs-vs-cats/train/"+sample)
- plt.imshow(image)

Build Model:-



- from keras.models import Sequential
- from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense, Activation, BatchNormalization
- model = Sequential()
- model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(IMAGE_WIDTH, IMAGE_HEIGHT, IMAGE_CHANNELS)))
- model.add(BatchNormalization())
- model.add(MaxPooling2D(pool_size=(2, 2)))
- model.add(Dropout(0.25))
- model.add(Conv2D(64, (3, 3), activation='relu'))
- model.add(BatchNormalization())
- model.add(MaxPooling2D(pool_size=(2, 2)))

- model.add(Dropout(0.25))
- model.add(Conv2D(128, (3, 3), activation='relu'))
- model.add(BatchNormalization())
- model.add(MaxPooling2D(pool_size=(2, 2)))
- model.add(Dropout(0.25))
- model.add(Flatten())
- model.add(Dense(512, activation='relu'))
- model.add(BatchNormalization())
- model.add(Dropout(0.5))
- model.add(Dense(2, activation='softmax'))
- model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
- model.summary()

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 126, 126, 32)      896
_________________________________________________________________
batch_normalization (BatchNo (None, 126, 126, 32)      128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 63, 63, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 61, 61, 64)        18496
_________________________________________________________________
batch_normalization_1 (Batch (None, 61, 61, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 30, 30, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 128)       73856
_________________________________________________________________
batch_normalization_2 (Batch (None, 28, 28, 128)       512
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 1024)              25691136
_________________________________________________________________
batch_normalization_3 (Batch (None, 1024)              4096
_________________________________________________________________
dropout (Dropout)            (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 2050
=================================================================
Total params: 25,791,426
Trainable params: 25,788,930
Non-trainable params: 2,496
```

Callbacks:-

- from keras.callbacks import EarlyStopping, ReduceLROnPlateau

Early Stop:-

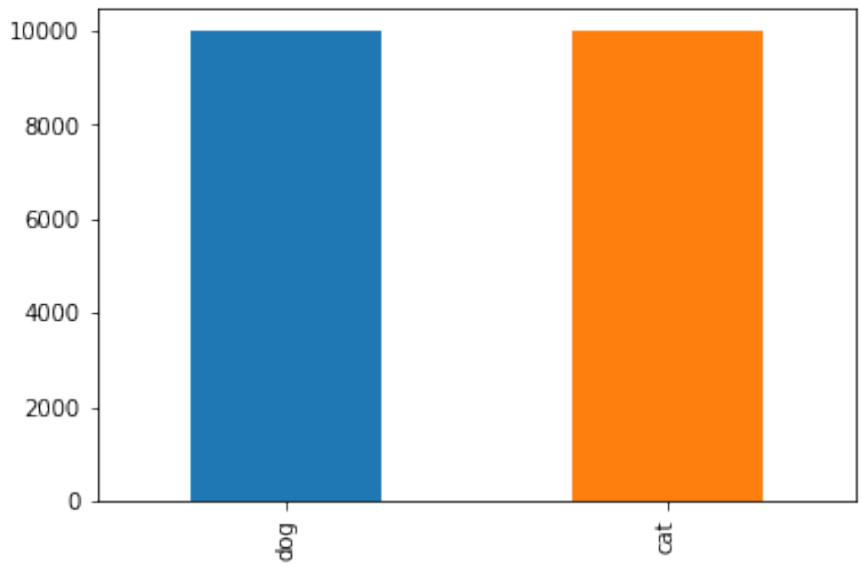
- earlystop = EarlyStopping(patience=10)

Learning Rate Reduction:-

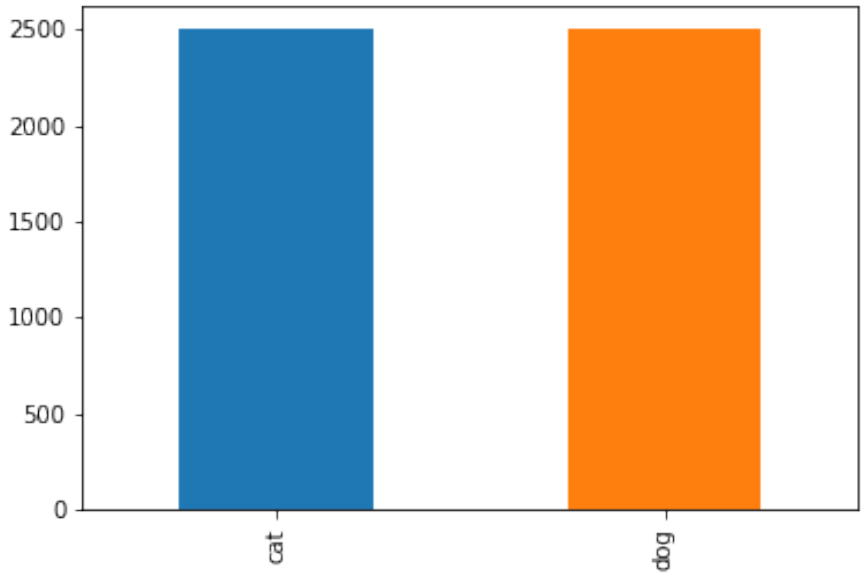
- learning_rate_reduction = ReduceLROnPlateau(monitor='val_acc',
  patience=2,
  verbose=1,
  factor=0.5,
  min_lr=0.00001)
- callbacks = [earlystop, learning_rate_reduction]

Prepare data:-

- df["category"] = df["category"].replace({0: 'cat', 1: 'dog'})
- train_df, validate_df = train_test_split(df, test_size=0.20, random_state=42)
- train_df = train_df.reset_index(drop=True)
- validate_df = validate_df.reset_index(drop=True)
- train_df['category'].value_counts().plot.bar()



- validate_df['category'].value_counts().plot.bar()



- total_train = train_df.shape[0]
- total_validate = validate_df.shape[0]
- batch_size=15

Training Generator:-

- train_datagen = ImageDataGenerator(rotation_range=15,rescale=1./255,
  shear_range=0.1, zoom_range=0.2,
  horizontal_flip=True, width_shift_range=0.1,
  height_shift_range=0.1)

- train_generator = train_datagen.flow_from_dataframe(train_df, "C:/Users/sa251/Downloads/dogs-vs-cats/train/",x_col='filename', y_col='category',target_size=IMAGE_SIZE, class_mode='categorical', batch_size=batch_size)

Validation Generator:-

- validation_datagen = ImageDataGenerator(rescale=1./255)

- validation_generator = validation_datagen.flow_from_dataframe(validate_df, "C:/Users/sa251/Downloads/dogs-vs-cats/train/", x_col='filename', y_col='category',target_size=IMAGE_SIZE, class_mode='categorical', batch_size=batch_size)

See how our generator work:-

- example_df = train_df.sample(n=1).reset_index(drop=True)

- example_generator = train_datagen.flow_from_dataframe(example_df, "C:/Users/sa251/Downloads/dogs-vs-cats/train/", x_col='filename', y_col='category',target_size=IMAGE_SIZE, class_mode='categorical')

- plt.figure(figsize=(12, 12))

- for i in range(0, 15):
    plt.subplot(5, 3, i+1)
    for X_batch, Y_batch in example_generator:
        image = X_batch[0]
        plt.imshow(image)
        break

- plt.tight_layout()

- plt.show()


0
25
50
75
100
125
0
50
100

Fit Model:-

- epochs=3 if FAST_RUN else 50
- history = model.fit_generator(train_generator, epochs=epochs, validation_data=validation_generator, validation_steps=total_validate//batch_size, steps_per_epoch=total_train//batch_size, callbacks=callbacks)

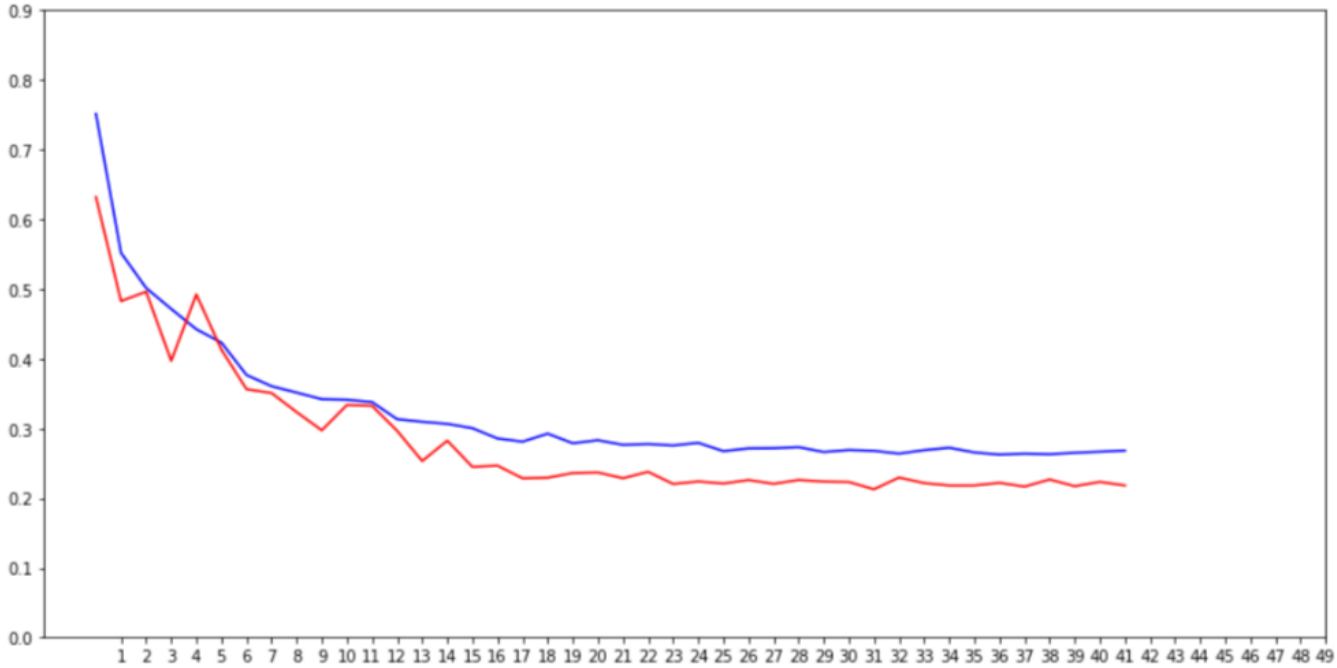
```
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/tensorflow/python/ops/math_op
s.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is deprecated and will be removed
in a future version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/50
1333/1333 [==============================] - 247s 186ms/step - loss: 0.7514 - acc: 0.6291 -
val_loss: 0.6320 - val_acc: 0.6603
Epoch 2/50
1333/1333 [==============================] - 199s 149ms/step - loss: 0.5521 - acc: 0.7241 -
val_loss: 0.4828 - val_acc: 0.7749
Epoch 3/50
1333/1333 [==============================] - 197s 148ms/step - loss: 0.5015 - acc: 0.7627 -
val_loss: 0.4962 - val_acc: 0.7535
Epoch 4/50
1333/1333 [==============================] - 197s 148ms/step - loss: 0.4718 - acc: 0.7775 -
val_loss: 0.3970 - val_acc: 0.8371
Epoch 5/50
1333/1333 [==============================] - 196s 147ms/step - loss: 0.4423 - acc: 0.8007 -
val_loss: 0.4924 - val_acc: 0.7840
Epoch 6/50
1333/1333 [==============================] - 196s 147ms/step - loss: 0.4233 - acc: 0.8096 -
val_loss: 0.4130 - val_acc: 0.8233

Epoch 00006: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
Epoch 7/50
1333/1333 [==============================] - 195s 146ms/step - loss: 0.3770 - acc: 0.8334 -
val_loss: 0.3563 - val_acc: 0.8548
Epoch 8/50
1333/1333 [==============================] - 195s 147ms/step - loss: 0.3606 - acc: 0.8439 -
val_loss: 0.3507 - val_acc: 0.8552
Epoch 9/50
1309/1333 [============================>.] - ETA: 3s - loss: 0.3524 - acc: 0.8490
```

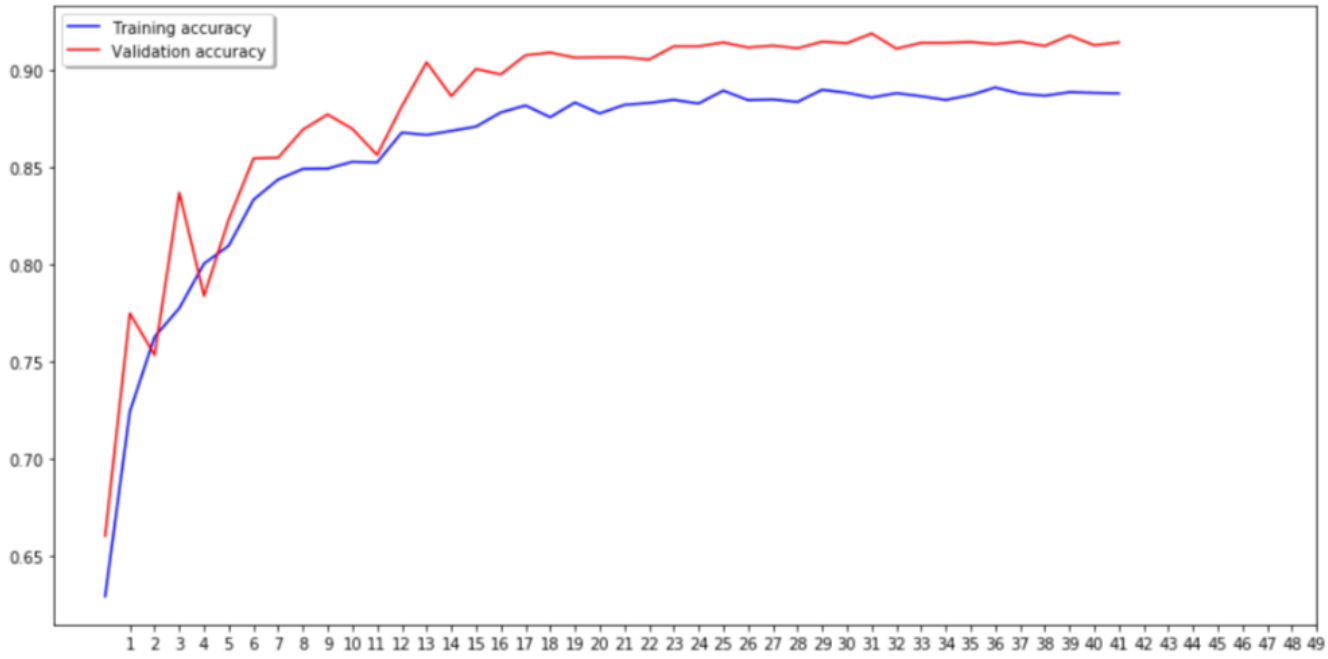
Save Model:-

- model.save_weights("saurabh1.h5")

Visualize Training:-

- fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 12))
- ax1.plot(history.history['loss'], color='b', label="Training loss")
- ax1.plot(history.history['val_loss'], color='r', label="validation loss")
- ax1.set_xticks(np.arange(1, epochs, 1))
- ax1.set_yticks(np.arange(0, 1, 0.1))
- ax2.plot(history.history['acc'], color='b', label="Training accuracy")
- ax2.plot(history.history['val_acc'], color='r',label="Validation accuracy")
- ax2.set_xticks(np.arange(1, epochs, 1))
- legend = plt.legend(loc='best', shadow=True)
- plt.tight_layout()
- plt.show()


0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49


Training accuracy
Validation accuracy
0.90
0.85
0.80
0.75
0.70
0.65
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49

Conclusion

“Image Classification” model developed to classify the celebrity image. I used Python Language with keras framework to implement this model. I used Convolutional Neural Networks (CNN) for image classification using images on Celebrity images data sets. This dataset used both training and testing purposes using CNN. It provides the accuracy rate of 65% on the training set and 14% on the test set. Images used in the training purpose are 64*64 and RGB color images. The future enhancement will focus on classifying the object detection in a video with better accuracy.

References

- https://keras.io/
- www.medium.com
- www.towardsdatascience.com
- https://www.youtube.com/

Git-hub link -:

https://github.com/saurabh8290/Mini-Project.git

Certificate