A Project Report on

Real Time Bare Skinned Image Filtering Using CNN

Submitted in partial fulfillment of the requirements for the award
of the degree of

Bachelor of Engineering

in

Computers

by

Pratik Jain(16102060)
Tina Shah(16102019)
Kshitija Shah(16102027)

Under the Guidance of

Prof.Jaya Gupta

Department of Computers
A.P. Shah Institute of Technology
G.B.Road,Kasarvadavli, Thane(W), Mumbai-400615
UNIVERSITY OF MUMBAI

Academic Year 2019-2020

Approval Sheet

This Project Report entitled ***" Real Time Bare Skinned Image Filtering Using CNN"*** Submitted by ***"Pratik Jain" (16102060),"Tina Shah" (16102019), "Kshitija Shah" (16102027)*** is approved for the partial fulfillment of the requirenment for the award of the degree of ***Bachelor of Engineering*** in ***Computers*** from ***University of Mumbai***.

Prof.Jaya Gupta
Guide

Prof. Sachin Malve
Head Deartment of Computers

Place:A.P.Shah Institute of Technology, Thane
Date:

CERTIFICATE

This is to certify that the project entitled ***"Real Time Bare Skinned Image Filtering Using CNN"*** submitted by ***"Pratik Jain" (16102060), "Tina Shah" (16102019), "Kshitija Shah" (16102027)*** for the partial fulfillment of the requirement for award of a degree ***Bachelor of Engineering*** in ***Computers***,to the University of Mumbai,is a bonafide work carried out during academic year 2019-2020.

Prof.Jaya Gupta
Guide

Prof. Sachin Malve
Head Department of Computers

Dr. Uttam D.Kolekar
Principal

External Examiner(s)

1.

2.

Place:A.P.Shah Institute of Technology, Thane
Date:

Declaration

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, We have adequately cited and referenced the original sources. We also declare that We have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

(Pratik Jain 16102060)

(Tina Shah 16102019)

(Kshitija Shah 16102027)

Date:

Abstract

Image classification is critical and significant research problems in computer vision applications such as facial expression classification, satellite image classification, and plant classification are based on images. This project proposes the image classification model applied for identifying the display of daunting pictures on the internet. The proposed model uses Convolutional Neural Network (CNN) to identify these images and filter them through different blocks of the network, so that it can be classified accurately. The model works on TensorFlow a user friendly platform, which provides different high levelled APIs (in Keras) which are used to build any basic model. It also permits us to add our own libraries. The input data for the model are images we collected online through various resources. We collected these images through scrapping using the Python library urllib.request. Our model will work as an extension to the web browser and will work on all websites when activated. The output of the proposed model is blurring of the images. This means that it will scan the entire web page and find all the daunting pictures present on that page, and then it will blur those images before they are loaded in front of the childrens eyes. This ensures protection from disturbing images and links, to the children at all cost.

Contents

List of Figures

Chapter 1

Introduction

We are living in the world of the Internet where Social Media and Browsing has become an important part of our life. Not only adults but even kids spend most of their time on the mobiles browsing for information and/or entertainment on different websites and social media platforms. As the use of Internet is growing tremendously, it has also become a very important medium for the advertisements of the product. It may happen that while browsing, children may come across advertisements containing nude images. There is a correct age to know things and seeing or getting exposed to nudity is not a good influence for young children. This exposure through the advertisements is not an appropriate knowledge exposure for them. While there are other applications available they might not always be free or full-proof to block these images.

Skin colour detection has been used in numerous computer vision applications like face detection, nudity recognition, hand gesture detection and person identification. But skin colour detection is a challenging task as the skin colour in an image is sensitive to various factors like illumination, camera characteristics, ethnicity, individual characteristics such as age, sex and body parts and other factors like makeup, hairstyle and glasses. Cues like shape and geometry can be used to build accurate face detection systems. These cues provide better accuracy in real time. These cues also save a lot of memory, as the pixels need not be saved for examination. So in our model we are using classification using the CNN which uses curves, edges and other features to identify and classify an image. As these are distinct geometry shapes and shows high resemblance in every human being the detection of these shapes and in turn humans are much more effective. We are training our model on both nude and non-nude images. In this study we have focused on the problem of developing an accurate and robust model for the nude images.

1.1 Objective

The model will offer child proof surfing on the internet without parent intervention. So that parents do not have to worry about their children coming across nude images at such an early age without knowing the actual meaning of it. That is it will monitor every page and filter all the images, hiding their details from the children and disabling their activation upon any click.Even if the people in the image is not completely nude, may it be just the upper or lower half of the persons body;thus ensuring guaranteed protection.

1.2 Literature Review

The Convolutional Neural Network (CNN) classifies the image regions as a collection of either skin or non-skin regions, it also classifies it based on the curves and edges it encounters in the image.Various approaches to skin modelling are used in the literature. Here we give a brief review of the neural network models for skin detection. The approach presented by Lee et al. employed a learning scheme based on the skin colour distribution of the image, using a neural network to learn and classify whether the input image contains skin exposure. Earlier work on nude identification focused on human skin detection, in which the idea is that greater amounts of detected skin would lead to higher probabilities of nudity within the image, hence characterizing the content as nude. Nevertheless, these approaches suffer with a high rate of false positives, especially in the context of beaches or practice of aquatic sports. An Tien Vo, Hai Son Tran, Thai Hoang Le suggested a method which involved ConvNetJS an open library which implements Deep Convolutional Neural Network. It is written in JavaScript and HTML with various supporting features for research purpose. It is an effective Neural Network, and is able to use CPU instead of GPU. The model proposed by them is an advertisement image classification system based on CNN consisting of 4 steps. It takes the image as an input from the webpage. It takes the image on the web browser by skin capturing and then saving that image. Then at the end classifying it by passing it through the CNN model. Our model is based on real time image disabling, so using this method would not work in our case.

Methods of data mining such as k-means were as well utilized for detection of skin and worth while results were obtained as demonstrated in the study by Hamid A. Jalab using acluster pixel model designed segmentation of skin under specifiable environmental conditions.This proposed model can overcome the changes in complex backgrounds and sensitivity to brightness conditions. Kakumanu et al. presented modeling and detection of skin and thisis recommended for further study. In conclusion, associated with the current issue, the most outstanding results is obtained using fuzzy system combined with support vector machine(FSFCSVM).

A residual learning framework to ease the training of networks that are substantially deeper than those used previously was put forth by Kaiming He, Xiangyu Zhang , Shaoqing Ren , Jian Sun . They explicitly reformulated the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. They provided comprehensive empirical evidence showing that these residual networks are easier to optimize,and can gain accuracy from considerably increased depth.

Apart from this the model proposed by Brown et al. contained two types of skin models used in the literature (Brown et al., 2001) viz., symmetric and asymmetric. Symmetric model uses a single classifier for both the classes whereas asymmetric model uses two separate classifiers for skin and non-skin pixels that are separately trained using respective features.Advantage of asymmetric skin classifier is that it increases the distances in certain skin related features between a positive (skin) and a negative (non-skin) image, with disadvantage of increased time complexity for training two classifiers. The neural classifiers used in the literature either uses a symmetric model with single neuron in the output layer or uses two separate neural networks (asymmetric model) for each of the skin and non-skin

classes. The novelty of our approach is that it has multiple convolutional that is hidden layer working on only skin coloured images containing humans.

1.3 Problem Definition

The main problem faced with CNN is training the model with the images it needs to identify.So in this study we train the model only on one type of image, which contains of exposed skin of humans, using a classifier. The classifier will differentiate the image based on its type.An image classified as positive by the classifier equals a nude or daunting image. Any other type of image will be filtered right through it. Thus this classifier makes the work easier, as we dont́ have to make two different classifiers for identification and filtering.The model then will be used in the creation of extension and accordingly the images will be blurred at runtime

1.4 Scope

This project proposes the image classification model applied for identifying the display of daunting pictures on the internet. In order to differentiate them from images containing humans from other images we train our model on nude images and all other classes of images .Our scope is to write our own CNN model which will capture the daunting images. Model will be able to classify whether the image is nude or non-nude for further processing. Even if the person in the image is not completely nude, i.e , even if the images are partially nude the model should be able to classify such type of images as nude images. This trained model will be used while creating the **chrome extension** to filter out the images that are nude or non-nude.If it finds nude image extension will blur the image. The user just need to download the chrome extension and it will start working without need on any setup in the users device.The Chrome extension will work for 24*7 with no charges and it will blur such exposed images on the website at runtime as the page gets loaded.

1.5 Technology Stack

Our technology is built on CNN model which is further used to create Chrome Extension which help us detect the nude pictures on the web pages.Our model mainly uses Convolutional Neural Network (CNN) for classifying the images correctly.To scrape images from the .txt file which have the links of required images,we have use python library urllib.request to store images on local machine.Images containing nudity should be separated from all other images in order to disable them. All this code is written in Google Colab, a platform which allows importing python libraries and uploading data from local computer or any other source.Keras is neural network library in python which makes it more way user friendly.We have used tensorflow.js (a python package) in order through convert the python code into a JavaScript format. Our internet only understands things in the .json format so we used the package tensorflw.js to convert our model written in python to JavaScript. Keras model is also required to retrieve data from a web page and pass it through our model. Therefore, Keras library worked as a crucial part of our project giving us flexibility to use our extension over the Chrome browser.The chrome Extension is stricly written in Javascript.

1.6 Benefits for Environment and society

This project would work as an excellent child proof surfing on the internet without us monitoring their usage 24*7. Not only the child would not come across extreme pictures and videos at such an early age where their minds are growing and has the most grasping power.But it will also help the parents to not waste their time and worry about their child getting exposed to nudity at such an early age.Thus, this will benefit the society as a whole in multiple ways the project mainly ensures protection of the children from a number of things while surfing on the internet.

1.7 Applications

We will try to put this technology on the web browser itself so that it works on all the websites. Thus being said we will try to provide it in the form of an extension which when activated would work for all the websites you visit while it is activated. So it has a wide range of applications, though it cannot work on all websites (e.g. porn websites) where there are multiple images and videos or links to them which cannot be blurred and disabled all at once. Our technology will still try to cover the maximum websites possible.The extensionwould work in Google Chrome as any other extension you may use or see. You will have to download and add it to the Google Chrome and give it permissions to scan the pages you visit while surfing on the internet.

Chapter 2

Project Design

2.1 Proposed System Architecture

Convolution Neural Network (CNN) is one of the most popular and effective technology for image recognition and image classification. The CNN classifier takes an image as an input,processes it and classifies it under certain categories. Our main aim is to disable the view of daunting images and get the link inactivated so that children do not get redirected to some irrelevant content which they are not ready to get exposed to at such an early age.We will first be training the CNN model (VGG16/VGG19) on bare skinned images as well as advertisements of any other type. During the prediction process the bare images will be giving an output of one, while all other type of images will have a value assigned to them greater than one. Once the model is trained with a good accuracy of prediction and filtering,a chrome extension will be created for the same, which will capture all the images on thew ebsites and if it comes across any such image, the image will be blurred. Hence this model would work as an excellent child proofsurfing on the internet without us monitoring their usage 24*7.

2.2 Flow of Models

First Step is the collection of dataset of both kind of nude and non-nude images.Images will be scrapped from the internet using the links of that particular image using concept of Beautiful Soap in python.Approximately 8600 images will be used in dataset for more accurate results.Next step is the classification of images, done by using Convolution neuralnetwork. The training of CNN would be done on two kinds of images positive and negative;positive being the nude images and negative being the images of any other type. Testing will be done for the same, so that we come to know how accurate the prediction is. Testing will also consists of both kind of images that is nude images and the images of advertisements.Extension will be created just like any other Google extension, which will blur and disablet he images on the website visited. The user just needs to download this Chrome extensionand enable it, after that all the working will be taken care of by the extension itself without any kind of parental control. This extension will work 24*7 and provide the safest browsing experience on the internet to the kids.

2.3 Activity Diagram

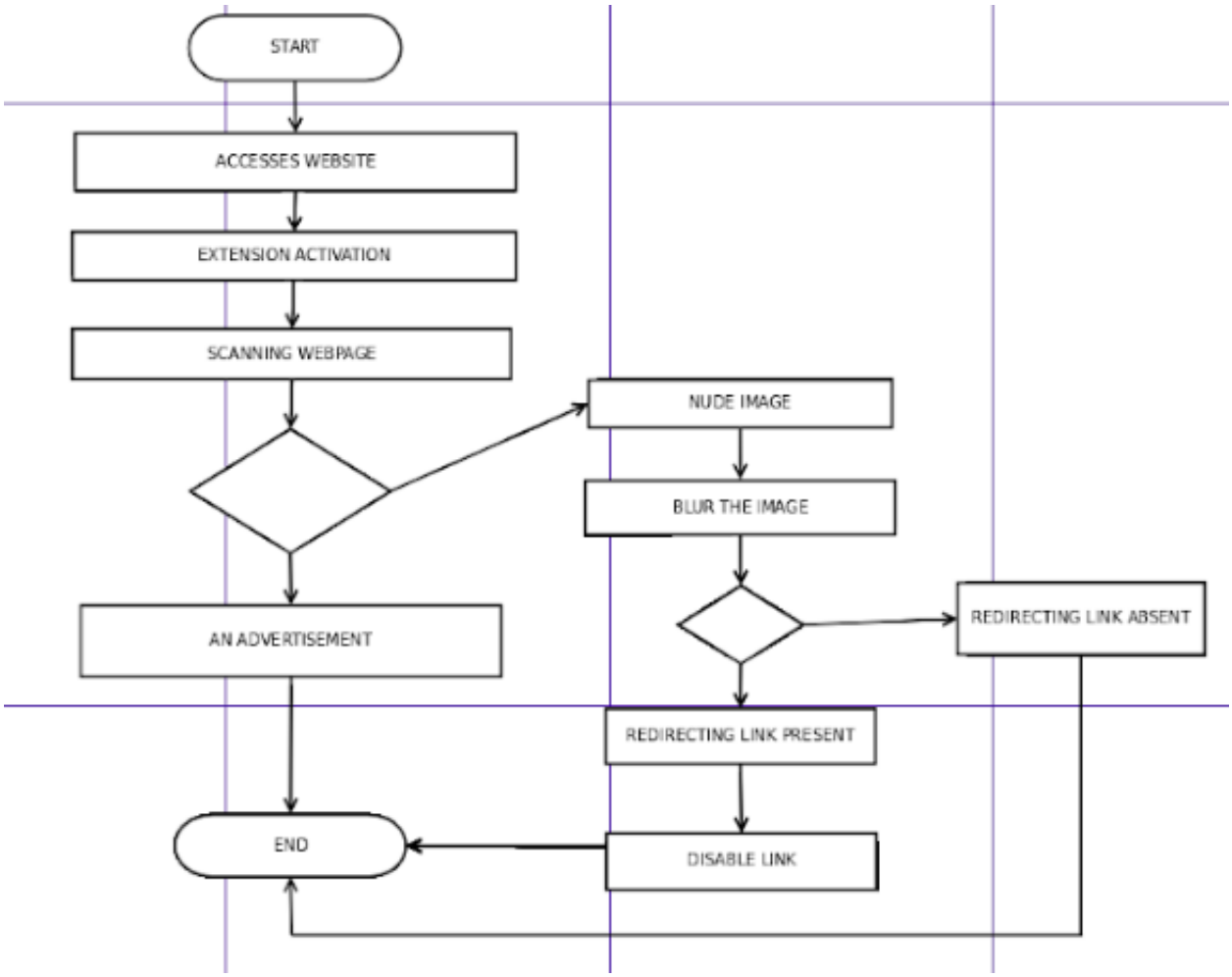

Figure 2.1: Activity Diagram

2.4 Use Case Diagram

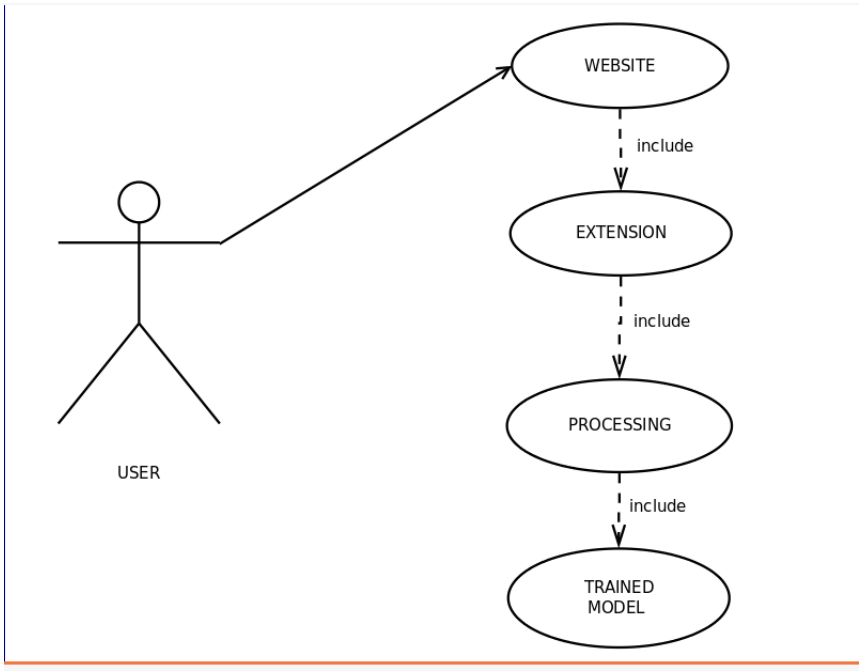

Figure 2.2: Use Case Diagram

Description of Use Case diagram

The above representation of our project is in the form of a Use Case diagram. This diagram represents the flow and the working of each model in the diagram. The diagram depicts that when the user accesses the website through the Google Chrome, the extensions working comes into the picture. The extension processes the entire webpage differentiating the images and blurring the positive images (nude images). This work is done on the basis of a trained model which the extension works upon to complete the desired action.

2.5 Models

The project is divided into different models as below:

2.5.1 Module 1 : Collection of Dataset

Convolution Neural Network (CNN) is one of the most popular and effective technology for image recognition and image classification.The CNN classifier takes the image as an input,processes it and classifies it under certain categories.As CNN takes image as an input,we need a good dataset on which cnn model will be trained. First objective was the collection of a good dataset which is the most important part while working in cnn.As this is a very sensitive thing ,the datasets are not readily available on the internet,hence we need to scrap images from the internet.The .txt files containing the links of both nude and non-nude images which are used for creation of dataset.Python library named urllib.request is used for scraping the images from the internet. Syntax for the same is urllib.request.urlretrieve(url,name), We need to pass 2 parameters, first is the variable url which has links and variable name which contains the string in which we want the image to be saved.The images are saved in the same directory where the code is present.Currently the dataset which is used for training the model has approximately 8000-9000 training images and 1000 testing images.

2.5.2 Module 2 : Classification Of Images Using CNN

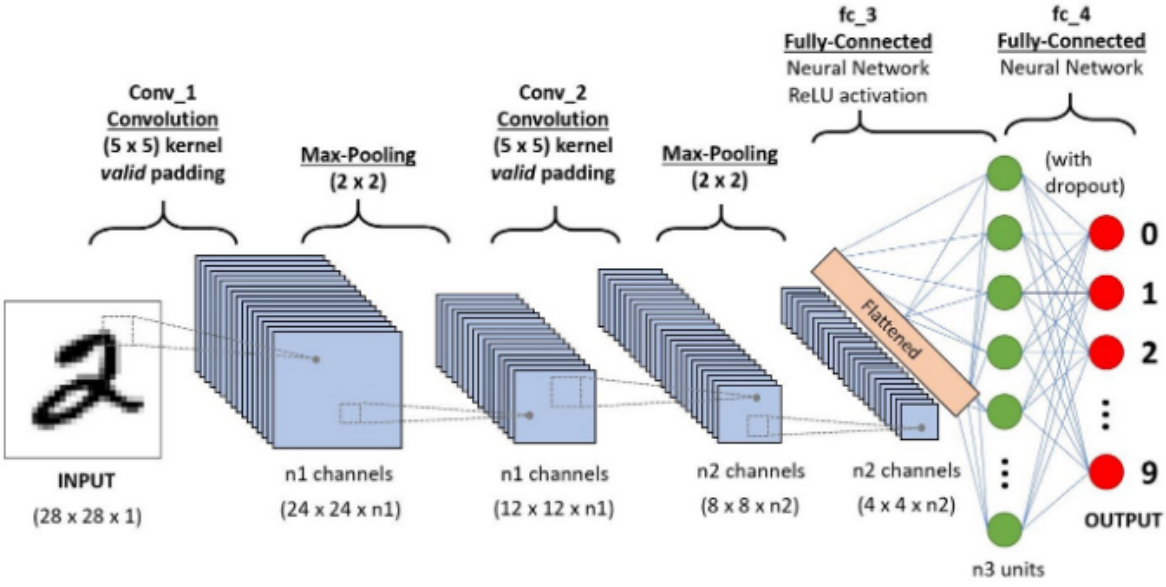

Figure 2.3: CNN model

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics. The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

Various types of CNN are as follows:

1] LeNet-5

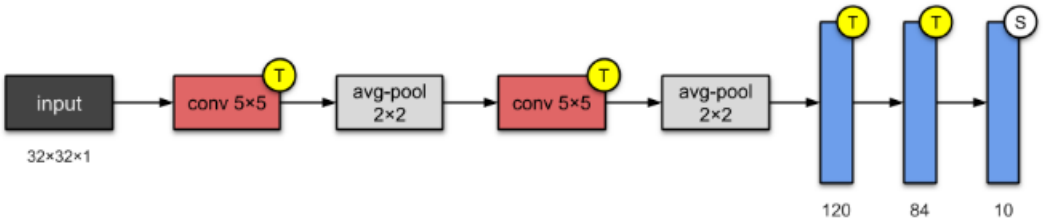

Figure 2.4: LeNet-5 Architecture

LeNet-5 is one of the simplest architectures. It has 2 convolutional and 3 fully-connected layers (hence 5 it is very common for the names of neural networks to be derived from the number of convolutional and fully connected layers that they have). The average-pooling layer as we know it now was called a sub-sampling layer and it had trainable weights (which isnt the current practice of designing CNNs nowadays). This architecture has about 60,000 parameters.

2]AlexNet

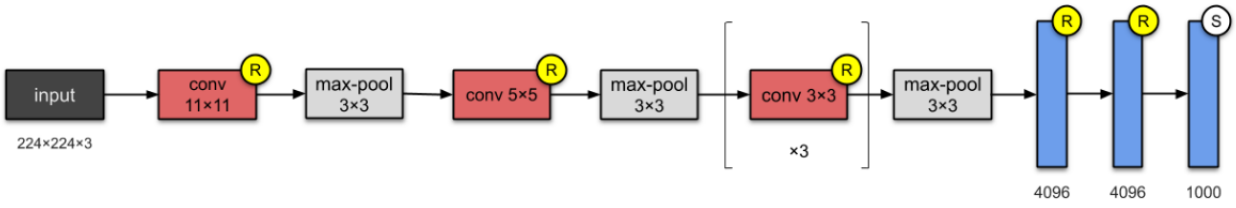

Figure 2.5: AlexNet Architecture

With 60M parameters, AlexNet has 8 layers 5 convolutional and 3 fully-connected. AlexNet just stacked a few more layers onto LeNet-5. At the point of publication, the authors pointed out that their architecture was one of the largest convolutional neural networks to date on the subsets of ImageNet.

3]VGG16 and VGG19

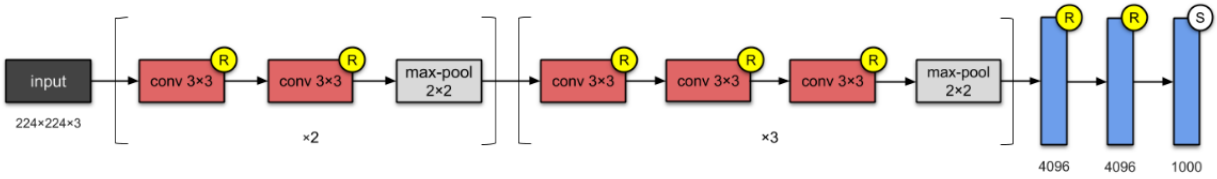

Figure 2.6: VGG16 Architecture

By now you wouldve already noticed that CNNs were starting to get deeper and deeper. This is because the most straightforward way of improving performance of deep neural networks is by increasing their size (Szegedy et. al). The folks at Visual Geometry Group (VGG) invented the VGG-16 which has 13 convolutional and 3 fully-connected layers, carrying with them the ReLU tradition from AlexNet. This network stacks more layers onto AlexNet, and use smaller size filters (22 and 33). It consists of 138M parameters and takes up about 500MB of storage space . They also designed a deeper variant, VGG-19.

4]Inception-v1

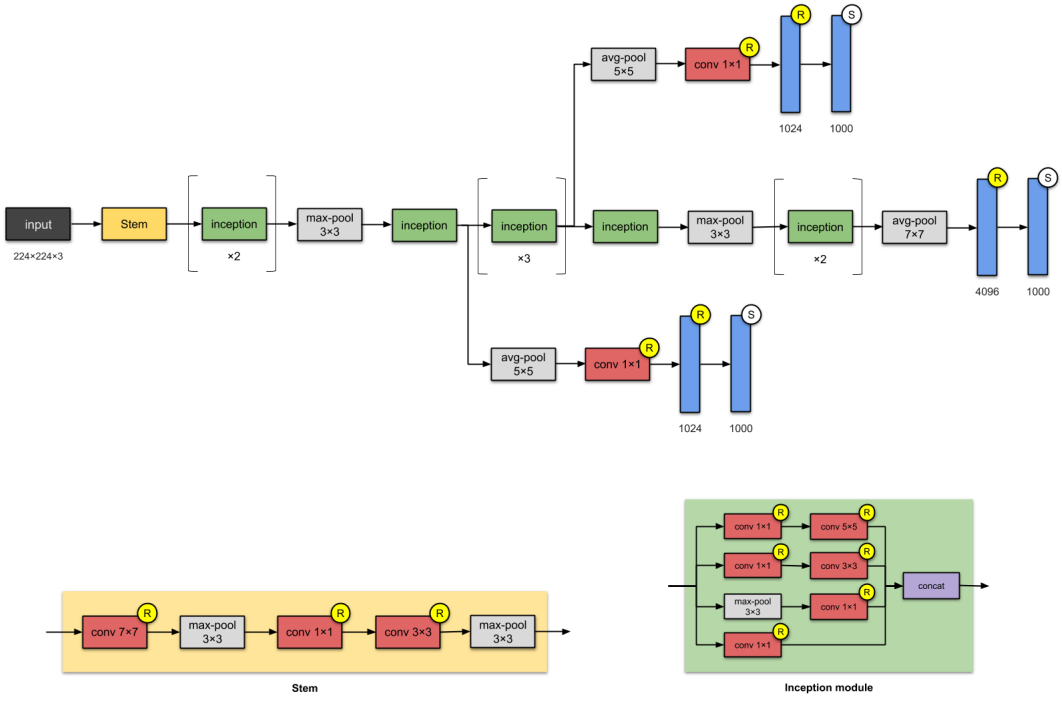

Figure 2.7: Inception-v1 Architecture

This 22-layer architecture with 5M parameters is called the Inception-v1. Here, the Network In Network approach is heavily used. This is done by means of Inception modules. The design of the architecture of an Inception module is a product of research on approximating sparse structures. Each module presents 3 ideas:

1.Having parallel towers of convolutions with different filters, followed by concatenation, captures different features at 11, 33 and 55, thereby clustering them. This idea is motivated by Arora et al. in the paper Provable bounds for learning some deep representations, suggesting a layer-by layer construction in which one should analyse the correlation statistics of the last layer and cluster them into groups of units with high correlation.

2.11 convolutions are used for dimensionality reduction to remove computational bottlenecks.

3.Due to the activation function from 11 convolution, its addition also adds nonlinearity.

5] ResNet-50

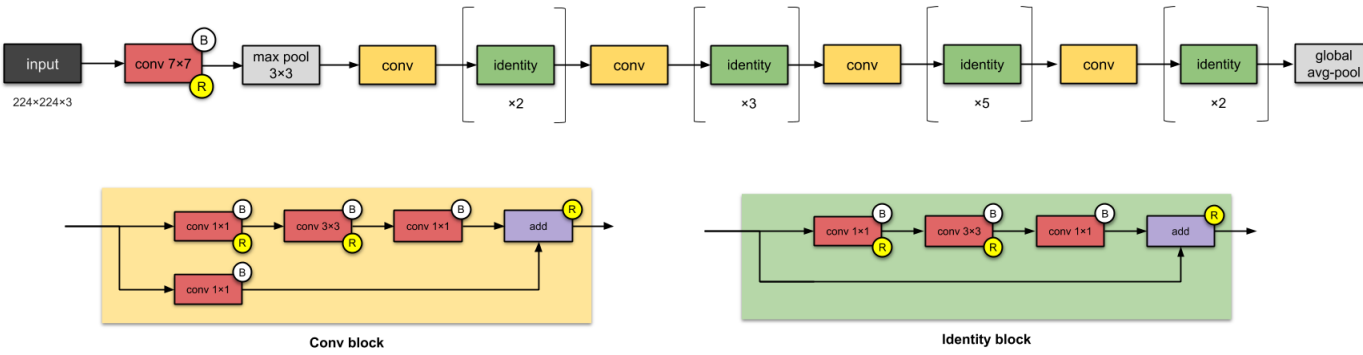

Figure 2.8: ResNet-50 Architecture

ResNet is one of the early adopters of batch normalisation (the batch norm paper authored by Ioffe and Szegedy was submitted to ICML in 2015). Shown above is ResNet-50, with 26M parameters. The basic building block for ResNets are the conv and identity blocks.

6] Xception

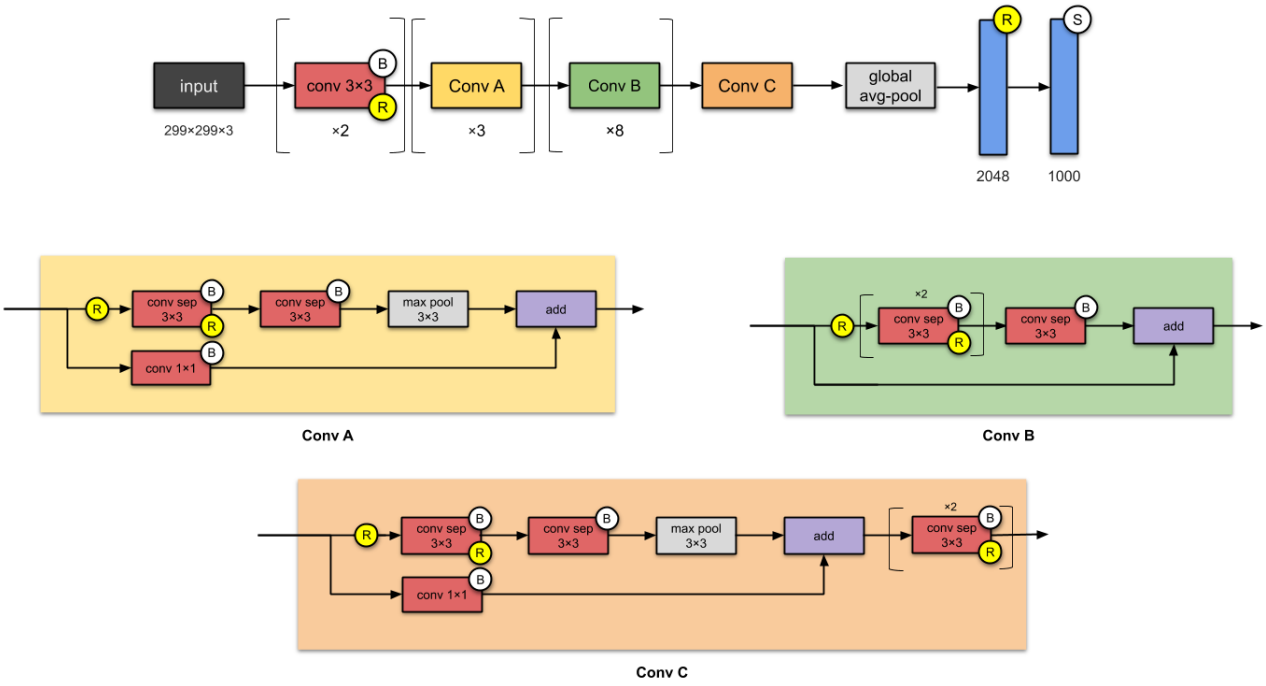

Figure 2.9: Xception Architecture

Xception is an adaptation from Inception, where the Inception modules have been replaced with depthwise separable convolutions. It has also roughly the same number of parameters as Inception-v1 (23M). Xception takes the Inception hypothesis to an eXtreme (hence the name).Firstly, cross-channel (or cross-feature map) correlations are captured by 11 convolutions. Consequently, spatial correlations within each channel are captured via the regular 33 or 55 convolutions. Taking this idea to an extreme means performing 11 to every channel, then performing a 33 to each output. This is identical to replacing the Inception module with depthwise separable convolutions.

After some research work,We came to a conclusion of using Vgg19 which works best on image classification.

2.5.3 Module 3 : Creation of Own Sequential Model

Keras Sequential model allows use to add Convolution layers according to our choice .We can add as many number of convolution layers as we need according to the requirements.We need three basic components to define a basic convolutional network:

- 1.The Convolution Layer
- 2.The Pooling Layer
- 3.The Outer Layer

1.The Convolution Layer

Suppose we have an image of size 6*6. We define a weight matrix which extracts certain features from the images.We have initialized the weight as a 3*3 matrix. This weight shall now run across the image such that all the pixels are covered at least once, to give a convolved output.The value is obtained by the adding the values obtained by element wise multiplication of the weight matrix and the highlighted 3*3 part of the input image. The 6*6 image

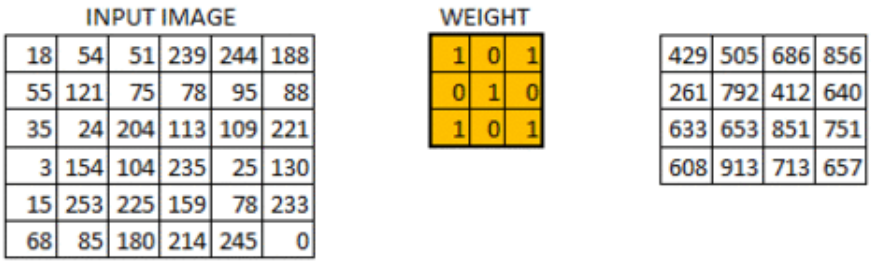

Figure 2.10: Convolution Layer Working

is now converted into a 4*4 image. The weight matrix behaves like a filter in an image extracting particular information from the original image matrix. A weight combination might be extracting edges, while another one might a particular color, while another one might just blur the unwanted noise.The weights are learnt such that the loss function is minimized similar to an MLP. Therefore weights are learnt to extract features from the original image which help the network in correct prediction. When we have multiple convolutional layers, the initial layer extract more generic features, while as the network gets deeper, the features extracted by the weight matrices are more and more complex and more suited to the problem at hand.

2.The pooling layer

Sometimes when the images are too large, we would need to reduce the number of trainable parameters. It is then desired to periodically introduce pooling layers between subsequent convolution layers. Pooling is done for the sole purpose of reducing the spatial size of the image. Pooling is done independently on each depth dimension, therefore the depth of the image remains unchanged. The most common form of pooling layer generally applied is

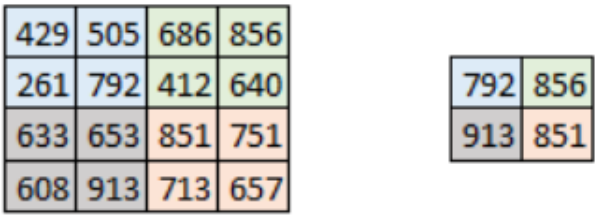

Figure 2.11: Pooling Layer

the max pooling. Here we have taken stride as 2, while pooling size also as 2. The max operation is applied to each depth dimension of the convolved output. As you can see, the 4*4 convolved output has become 2*2 after the max pooling operation.

3.The output Layer

After multiple layers of convolution and padding, we would need the output in the form of a class. The convolution and pooling layers would only be able to extract features and reduce the number of parameters from the original images. However, to generate the final output we need to apply a fully connected layer to generate an output equal to the number of classes we need. It becomes tough to reach that number with just the convolution layers. Convolution layers generate 3D activation maps while we just need the output as whether or not an image belongs to a particular class. The output layer has a loss function like categorical cross-entropy, to compute the error in prediction. Once the forward pass is complete the backpropagation begins to update the weight and biases for error and loss reduction.

2.5.4 Module 4 : Activation Functions and Optimizers Analysis

Activation Functions

Activation functions are really important for an Artificial Neural Network to learn and make sense of things which are really complicated. They introduce non-linear properties to our Neural Network. Their main purpose is to convert an input signal of a node to an output signal. That output signal now is used as an input in the next layer in the stack. Specifically in an Artificial Neural Network we do the sum of products of inputs(X) and their corresponding Weights (W) and apply an Activation function f(x) to it to get the output of that layer and feed it as an input to the next layer. A Neural Network without Activation function would simply be a Linear regression Model, which has limited power and without activation function our Neural network would not be able to learn and model other complicated kinds of data such as images, videos , audio , speech etc. That is why we use Artificial Neural network techniques such as Deep learning to make sense of something complicated , high dimensional, non-linear -big datasets, where the model has lots and lots of hidden layers in between and has a very complicated architecture which helps us to make sense and extract knowledge form such complicated big datasets.
Types of Activation functions -

1]Sigmoid or Logistic

sigmoid(x) = 1 / (1 + exp(-x)).
Applies the sigmoid activation function. For small values (-5), sigmoid returns a value close to zero, and for large values (5)$the result of the function gets close to 1. Sigmoid is equivalent to a 2-element Softmax, where the second element is assumed to be zero. The sigmoid function always returns a val$

2]Tanh Hyperbolic tangent

f(x) = 1 exp(-2x) / 1 + exp(-2x).
Now its output is zero centred because its range in between -1 to 1 i.e -1 lt; output lt; 1 . Hence optimization is easier in this method hence in practice it is always preferred over Sigmoid function . But still it suffers from Vanishing gradient problem

3]ReLu -Rectified linear units

Rectified Linear units: It has become very popular in the past couple of years. It was recently proved that it had 6 times improvement in convergencefrom Tanh function. Its just R(x) = max(0,x) i.e if x lt; 0 , R(x) = 0 and if x gt;= 0 , R(x) = x. Hence as seeing the mathematical form of this function we can see that it is very simple and efficient . A lot of times in Machine learning and computer science we notice that most simple and consistent techniques and methods are only preferred and are best. Hence it avoids and rectifies vanishing gradient problem. Almost all deep learning Models use ReLu nowadays. But its limitation is that it should only be used within Hidden layers of a Neural Network Model. Another problem with ReLu is that some gradients can be fragile during training

and can die. It can cause a weight update which might result in no activation of any data point, resulting into Dead Neurons.

4]Elu:

The exponential linear activation: x if x ¿ 0 and alpha * (exp(x)-1) if x ¡ 0.
exponential linear unit (ELU) which speeds up learning in deep neural networks and leads to higher classification accuracies. Like rectified linear units (ReLUs), leaky ReLUs (LReLUs) and parametrized ReLUs (PReLUs), ELUs alleviate the vanishing gradient problem via the identity for positive values. However, ELUs have improved learning characteristics compared to the units with other activation functions. In contrast to ReLUs, ELUs have negative values which allows them to push mean unit activations closer to zero like batch normalization but with lower computational complexity. Mean shifts toward zero speed up learning by bringing the normal gradient closer to the unit natural gradient because of a reduced bias shift effect.

5]Selu

The Scaled Exponential Linear Unit (SELU) activation function is defined as:

- x greater then 0: return scale * x
- x less then 0: return scale * alpha * (exp(x) - 1)

where alpha and scale are pre-defined constants (alpha=1.67326324 and scale=1.05070098). The values of alpha and scale are chosen so that the mean and variance of the inputs are preserved between two consecutive layers as long as the weights are initialized correctly and the number of input units is "large enough".

6]Hard Sigmoid

Hard sigmoid is a generalization of the sigmoid activation function. Since its a generalization and almost linear its much faster and cheaper to compute than an ordinary sigmoid activation function. Hard sigmoid is the default recurrent activation for LSTM models.

7]Softsign

softsign(x) = x / (abs(x) + 1)
As an alternative to hyperbolic tangent, softsign is an activation function for neural networks. Even though tanh and softsign functions are closely related, tanh converges exponentially whereas softsign converges polynomially. Even though softsign appears in literature, it would not be adopted in practice as much as tanh.

8]Softmax

Softmax converts a real vector to a vector of categorical probabilities.The elements of the output vector are in range (0, 1) and sum to 1.Each vector is handled independently. The axis argument sets which axis of the input the function is applied along.Softmax is often used as the activation for the last layer of a classification network because the result could be interpreted as a probability distribution.The softmax of each vector x is computed as exp(x) / tf.reducesum(exp(x)).The input values in are the log-odds of the resulting probability.

9]Softplus

softplus(x) = log(exp(x) + 1)
Softmax function, a wonderful activation function that turns numbers aka logits into probabilities that sum to one. Softmax function outputs a vector that represents the probability distributions of a list of potential outcomes.

10]Linear

It takes the inputs, multiplied by the weights for each neuron, and creates an output signal proportional to the input. In one sense, a linear function is better than a step function because it allows multiple outputs, not just yes and no.

Optimizers

Optimizers tie together the loss function and model parameters by updating the model in response to the output of the loss function. In simpler terms, optimizers shape and mould your model into its most accurate possible form by editing the weights. The loss function is the guide to the terrain, telling the optimizer when its moving in the right or wrong direction. It is impossible to know what your models weights should be right from the start. But with some trial and error based on the loss function, you can end up getting the right and the best values eventually. Here are few of the optimizers that are present in todays Machine Learning world:

1]Stochastic Gradient Descent

Instead of calculating the gradients for all of your training examples on every pass of gradient descent, its sometimes more efficient to only use a subset of the training examples each time.Stochastic gradient descentis an implementation that either uses batches of examples at a time or random examples on each pass. We specifically havent included the formal functions for the concepts in this post because were trying to explain things intuitively. For more insight into the math involved and a more technical analysis,this walk through guide with Excel examples is helpful.

2]Adagrad

Adagrad adapts the learning rate specifically to individual features: that means that some of the weights in your dataset will have different learning rates than others. This works really well for sparse datasets where a lot of input examples are missing. Adagrad has a major

issue though: the adaptive learning rate tends to get really small over time. Some other optimizers below seek to eliminate this problem.

3]RMSprop

RMSprop is a special version of Adagrad developed by Professor Geoffrey Hinton inhis neural nets class. Instead of letting all of the gradients accumulate for momentum, it only accumulates gradients in a fixed window. RMSprop is similar to Adaprop, which is another optimizer that seeks to solve some of the issues that Adagrad leaves open.

4]Adam

Adamstands for adaptive moment estimation, and is another way of using past gradients to calculate current gradients. Adam also utilizes the concept ofmomentumby adding fractions of previous gradients to the current one. This optimizer has become pretty widespread, and is practically accepted for use in training neural nets.
Its easy to get lost in the complexity of some of these new optimizers. Just remember that they all have the same goal: minimizing our loss function. Even the most complex ways of doing that are simple at their core. But for our model the best that works is SGD along with sigmoid. So we have used that in our model as well. This gives us a accuracy of 87.29%.

5]Adadelta

Adadelta is a more robust extension of Adagrad that adapts learning rates based on a moving window of gradient updates, instead of accumulating all past gradients. This way, Adadelta continues learning even when many updates have been done. Compared to Adagrad, in the original version of Adadelta you don't have to set an initial learning rate. In this version, initial learning rate can be set, as in most other Keras optimizers.

6]Adamax

It is a variant of Adam based on the infinity norm. Default parameters follow those provided in the paper. Adamax is sometimes superior to adam, specially in models with embeddings.

7]Nadam

Much like Adam is essentially RMSprop with momentum, Nadam is Adam with Nesterov momentum.

All the combinations of activation functions and optimizers are applied on the model created using sequential layer and best will be choosen out of all the combination which will give best accuracy.

2.5.5 Module 5 : Creation Of Chrome Extensions

Extensions are small software programs that customize the browsing experience. They enable users to tailor Chrome functionality and behavior to individual needs or preferences. They are built on web technologies such as HTML, JavaScript, and CSS.An extension must fulfill a single purpose that is narrowly defined and easy to understand. A single extension can include multiple components and a range of functionality, as long as everything contributes towards a common purpose.

First process for chrome extension is the extraction of src of the images from the websites you are currently browsing.Once we get all the src of the images,we need to load the images which has it's information like size we need to reshape the size of image in 200*200 size. As the model created using cnn has the training images of size 200*200 or it will create an error.Then it is passed to main cnn model for classification.If the image is nude value ¿0.5 will be predicted and if non-nude ¡0.5 value will be predicted.If value is ¿0.5, the image will get blured on the website.

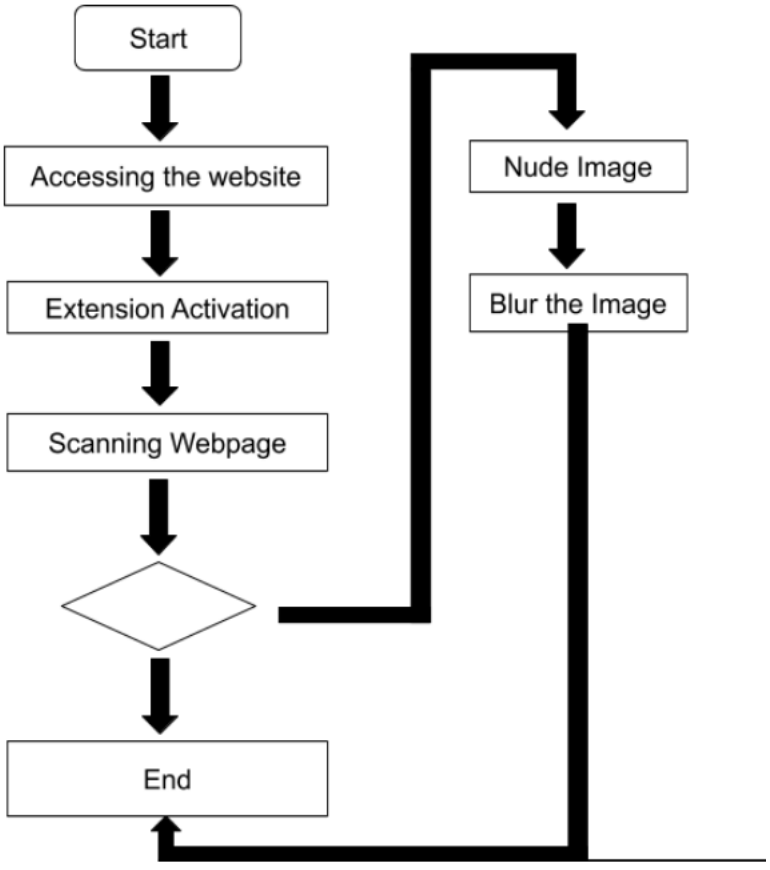

Figure 2.12: Google Chrome Extension Working

Chapter 3

Project Implementation

3.0.1 Module 1 : Collection of Dataset

Training is done on a dataset of 8000-9000 images for more accurate and correct results. The dataset of 8000 images was not easy to find for one reason; it was not readily available on any website like Kaggle for example which is a source of various types of datasets. So now we had to prepare our own dataset. For this purpose we used a concept known as BeautifulSoup or more commonly known as Scrapping. Scrapping basically is downloading all the images from a page and saving it into our computer. The name scrapping suggests that we do not download each and every content form the web page, but only scrap out the content we need. To achieve the mentioned task we wrote a code which runs in a loop format. It scans each and every link from a text file, visits that link and download whatever image is present on that link. We found this list of links from a GitHub account. This collection of data takes several weeks as, the amount of data is huge. Also one problem you might face is the blank links. As these links are random, it is possible that the image that was previously present is not available any more. When the program encounters such a problem it stops over there and we have to remove that specific link from the text folder in order to scan the rest of the image links. So it requires you attention on a periodic basis even though it is a self-running program. Now that the data is collected we had to train the model from the same.The code used to scrap images is as follows:

```
import urllib.request



def scraping(url,numb):
        print(url)
        name="nude"+str(numb)+".jpg"
        urllib.request.urlretrieve(url,name)

i=1
while(True):
        file1 = open("myfile2.txt","r")
        for x in file1:
                scraping(x,i)
                i+=1
file1.close()
```

3.0.2 Module 2 : Classification Of Images Using CNN

Out of different CNN algorithms,VGG19 is used in the project which works best in case of training model having images. It consists of 19 layers. When we trained the 8000-9000 images using this model we got an accuracy of 84.28% on 10 epochs and 85..39% on 15 epoch.Details of common parameters used are as follows:

- Optimizer = SGD
- Activation=sigmoid
- Training images = 5604
- Testing images = 999

Code for the same is as follows:

```
from os import listdir
from numpy import asarray
from numpy import save
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array

folder='/content/drive/My Drive/major projects/final/training/'
photos,labels=list(),list()
i=1
for file in listdir(folder):
  print(str(i)+" "+file)
  output=0.0
  if(file.startswith('nude')):
    output=1.0
  photo=load_img(folder+file,target_size=(200,200))
  photo=img_to_array(photo)
  photos.append(photo)
  labels.append(output)
  i+=1
X_train=asarray(photos)
Y_train=asarray(labels)
#print(X_train)
#print(Y_train)
```

```
from os import listdir
from numpy import asarray
from numpy import save
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
folder="/content/drive/My Drive/major projects/final/testing/"
photos,labels=list(),list()
i=1
for file in listdir(folder):
  print(str(i)+" "+file)
  output=0.0
  if(file.startswith('nude')):
    output=1.0
  photo=load_img(folder+file,target_size=(200,200))
  photo=img_to_array(photo)
  photos.append(photo)
  labels.append(output)
  i+=1
X_test=asarray(photos)
Y_test=asarray(labels)
#print(X_test)
#print(Y_test)
```

Vgg19 model

```
import sys
from matplotlib import pyplot
from keras.utils import to_categorical
from keras.applications.vgg19 import VGG19
from keras.models import Model
from keras.layers import Dense
from keras.layers import Flatten
from keras.optimizers import SGD

model=VGG19(include_top=False,input_shape=(200,200,3))
for layer in model.layers:
  layer.trainable=False
flat1=Flatten()(model.layers[-1].output)
class1=Dense(500,activation='relu',kernel_initializer="he_uniform")(flat1)
class2=Dense(500,activation='relu',kernel_initializer="he_uniform")(class1)
class3=Dense(500,activation='relu',kernel_initializer="he_uniform")(class2)
class4=Dense(500,activation='relu',kernel_initializer="he_uniform")(class3)
output=Dense(1,activation='sigmoid')(class4)
model=Model(inputs=model.inputs,outputs=output)
```

Compilation of model

```
opt=SGD(lr=0.001,momentum=0.9)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train,Y_train,validation_data=(X_test,Y_test),batch_size=128,epochs=10,verbose=1)
#model.save_weights("weights.h5")
#model.save("final.h5")
```

output with 10 epochs

```
Epoch 6/10
5604/5604 [==============================] - 16s 3ms/step - loss: 0.0050 - acc: 0.9993 - val_los
s: 0.6693 - val_acc: 0.8529
Epoch 7/10
5604/5604 [==============================] - 16s 3ms/step - loss: 0.0047 - acc: 0.9995 - val_los
s: 0.7052 - val_acc: 0.8468
Epoch 8/10
5604/5604 [==============================] - 16s 3ms/step - loss: 0.0050 - acc: 0.9991 - val_los
s: 0.6670 - val_acc: 0.8549
Epoch 9/10
5604/5604 [==============================] - 16s 3ms/step - loss: 0.0046 - acc: 0.9993 - val_los
s: 0.6720 - val_acc: 0.8559
Epoch 10/10
5604/5604 [==============================] - 16s 3ms/step - loss: 0.0058 - acc: 0.9993 - val_los
s: 0.7326 - val_acc: 0.8428
```

output with 15 epochs

```
Epoch 12/15
5604/5604 [==============================] - 16s 3ms/step - loss: 0.0064 - acc: 0.9995 - val_los
s: 0.6784 - val_acc: 0.8579
Epoch 13/15
5604/5604 [==============================] - 16s 3ms/step - loss: 0.0064 - acc: 0.9995 - val_los
s: 0.7408 - val_acc: 0.8438
Epoch 14/15
5604/5604 [==============================] - 16s 3ms/step - loss: 0.0064 - acc: 0.9995 - val_los
s: 0.6634 - val_acc: 0.8619
Epoch 15/15
5604/5604 [==============================] - 16s 3ms/step - loss: 0.0065 - acc: 0.9993 - val_los
s: 0.7027 - val_acc: 0.8539
```

Prediction of saved mode

```
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.models import load_model
from os import listdir
path="/content/drive/My Drive/major projects/nude and non-nude/test/"
def load_image(filename):
  img=load_img(filename,target_size=(200,200))
  img=img_to_array(img)
  img=img.reshape(1,200,200,3)
  img=img.astype('float32')
  return img

def run_example():
  for file in listdir(path):
    z=path+file
    print(z)
    img=load_image(z)
    model=load_model("/content/drive/My Drive/major projects/final.h5")
    result=model.predict(img)
    print(result[0])

run_example()
```

```
/content/drive/My Drive/major projects/nude and non-nude/test/No444.jpg
[0.]
/content/drive/My Drive/major projects/nude and non-nude/test/Yes486.jpg
[0.9999287]
/content/drive/My Drive/major projects/nude and non-nude/test/No498.jpg
[2.7357288e-05]
/content/drive/My Drive/major projects/nude and non-nude/test/Yes500.jpg
[0.99979466]
/content/drive/My Drive/major projects/nude and non-nude/test/No402.jpg
[0.47208062]
/content/drive/My Drive/major projects/nude and non-nude/test/No460.jpg
[0.]
/content/drive/My Drive/major projects/nude and non-nude/test/No462.jpg
[0.00023309]
/content/drive/My Drive/major projects/nude and non-nude/test/No480.jpg
[9.990355e-08]
/content/drive/My Drive/major projects/nude and non-nude/test/No458.jpg
[0.]
/content/drive/My Drive/major projects/nude and non-nude/test/No484.jpg
[0.00032475]
```

3.0.3 Module 3 : Creation of Own Sequential Model

The proposed model uses Convolutional Neural Network (CNN) to identify these images and filter them through different blocks of the network, so that it can be classified accurately. Training is done using the CNN model to identify the bare images and the images of advertisement.Our model proposes the use of VGG16/VGG19 model. Our CNN model consists of total 26 layers.There are 15 convolutional layer,5 max pooling layers and 7 fully connected layers having neurons in decreasing manner in each layer.Keeping the few same like optimizer,activation function,training images,testing images as used in VGG19 model.

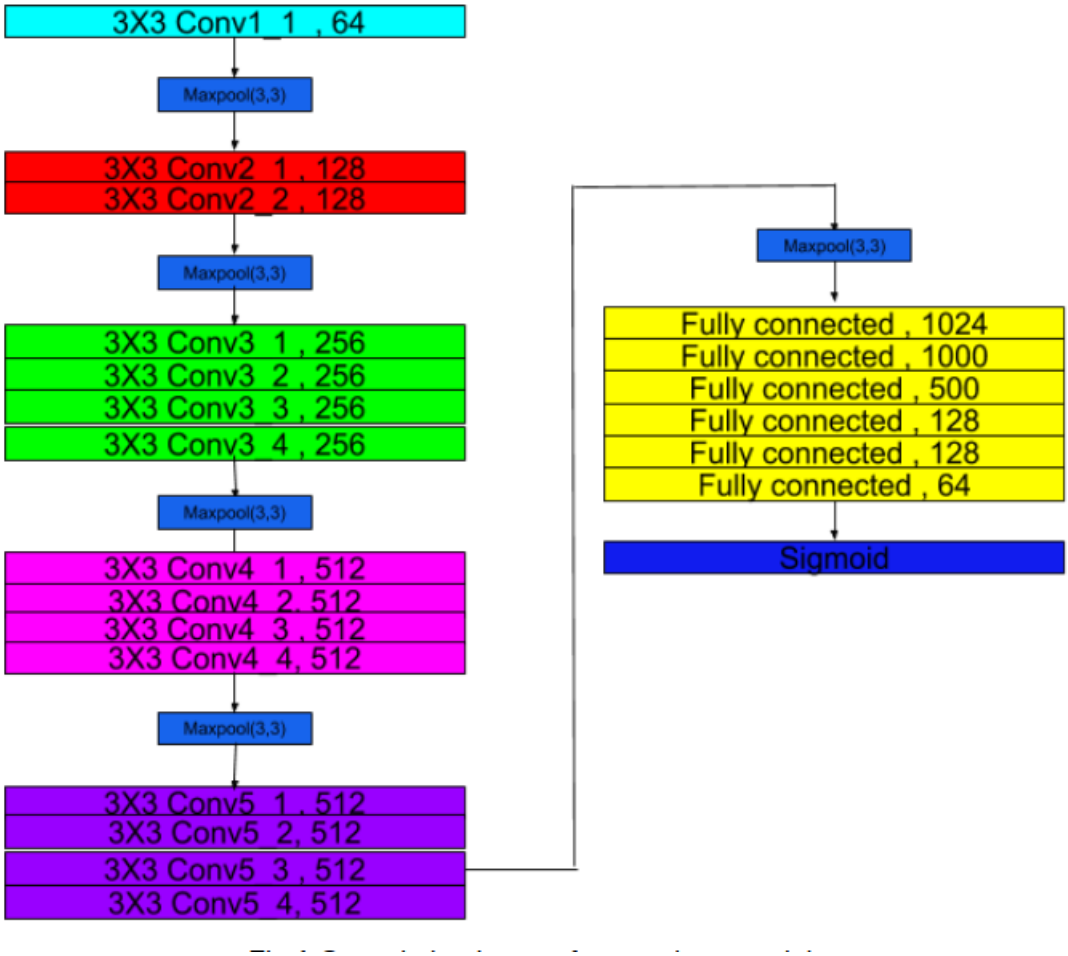

Figure 3.1: Flow of CNN Model

Code of sequential model with layers

```
import sys
from matplotlib import pyplot
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.optimizers import SGD

model = Sequential()
model.add(Conv2D(input_shape=(200,200,3),filters=64,kernel_size=(3,3),padding="same", activatio
n="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))


model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))




model.add(Flatten())

model.add(Dense(units=1024,activation="relu"))
model.add(Dense(units=1000,activation="relu"))
model.add(Dense(units=500,activation="relu"))
model.add(Dense(units=128,activation="relu"))
model.add(Dense(units=128,activation="relu"))
model.add(Dense(units=64,activation="relu"))
model.add(Dense(units=1, activation="sigmoid"))
```

Compilation of sequential model

```
opt=SGD(lr=0.001,momentum=0.9)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train,Y_train,validation_data=(X_test,Y_test),batch_size=128,epochs=10,verbose=1)
#model.save_weights("weights.h5")
#model.save("final.h5")
```

output with 10 epochs

```
Epoch 6/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.5193 - acc: 0.7557 - val_los
s: 0.4426 - val_acc: 0.8148
Epoch 7/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.5123 - acc: 0.7579 - val_los
s: 0.5155 - val_acc: 0.7818
Epoch 8/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.4939 - acc: 0.7744 - val_los
s: 0.6098 - val_acc: 0.6997
Epoch 9/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.4969 - acc: 0.7616 - val_los
s: 0.3801 - val_acc: 0.8519
Epoch 10/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.5082 - acc: 0.7595 - val_los
s: 0.4281 - val_acc: 0.8639
```

output with 15 epochs

```
Epoch 12/15
5604/5604 [==============================] - 38s 7ms/step - loss: 0.4857 - acc: 0.7741 - val_los
s: 0.3581 - val_acc: 0.8629
Epoch 13/15
5604/5604 [==============================] - 38s 7ms/step - loss: 0.4804 - acc: 0.7741 - val_los
s: 0.3733 - val_acc: 0.8539
Epoch 14/15
5604/5604 [==============================] - 38s 7ms/step - loss: 0.4822 - acc: 0.7703 - val_los
s: 0.5174 - val_acc: 0.7808
Epoch 15/15
5604/5604 [==============================] - 38s 7ms/step - loss: 0.4780 - acc: 0.7784 - val_los
s: 0.3424 - val_acc: 0.8729
```

3.0.4 Module 4 : Activation Functions and Optimizers Analysis

As there are 7 optimizers and 7 activation functions, we cannot predict will would give the best accuracy.When we compare between the VGG19 and Sequential model, the sequential model gives the best output,so we will try different combinations of optimizers and activation functions to get the best results.

Activation funtion = tanh with all combinations of optimizers

```
import sys
from matplotlib import pyplot
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.optimizers import SGD


model = Sequential()
model.add(Conv2D(input_shape=(200,200,3),filters=64,kernel_size=(3,3),padding="same", activatio
n="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))


model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Flatten())

model.add(Dense(units=1024,activation="relu"))
model.add(Dense(units=1000,activation="relu"))
model.add(Dense(units=500,activation="relu"))
model.add(Dense(units=128,activation="relu"))
model.add(Dense(units=128,activation="relu"))
model.add(Dense(units=64,activation="relu"))
model.add(Dense(units=1, activation="tanh"))
```

```
opt=SGD(lr=0.001,momentum=0.9)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train,Y_train,validation_data=(X_test,Y_test),batch_size=128,epochs=10,verbose=1)
#model.save_weights("weights.h5")
#model.save("final.h5")
```

```
Epoch 6/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.5253 - acc: 0.7496 - val_los
s: 0.7279 - val_acc: 0.6156
Epoch 7/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.5418 - acc: 0.7364 - val_los
s: 0.5505 - val_acc: 0.7748
Epoch 8/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.5165 - acc: 0.7532 - val_los
s: 0.4548 - val_acc: 0.8198
Epoch 9/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.4987 - acc: 0.7643 - val_los
s: 0.4367 - val_acc: 0.8248
Epoch 10/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.4983 - acc: 0.7598 - val_los
s: 0.4469 - val_acc: 0.8148
```

```
opt2=RMSprop(lr=0.001, rho=0.9)
model.compile(optimizer=opt2, loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train,Y_train,validation_data=(X_test,Y_test),batch_size=128,epochs=10,verbose=1)
#model.save_weights("weights.h5")
#model.save("final.h5")
```

```
Epoch 6/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
Epoch 7/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
Epoch 8/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
Epoch 9/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
Epoch 10/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
```

Activation funtion = elu with all combinations of optimizers

```
import sys
from matplotlib import pyplot
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.optimizers import SGD


model = Sequential()
model.add(Conv2D(input_shape=(200,200,3),filters=64,kernel_size=(3,3),padding="same", activatio
n="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))


model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Flatten())

model.add(Dense(units=1024,activation="relu"))
model.add(Dense(units=1000,activation="relu"))
model.add(Dense(units=500,activation="relu"))
model.add(Dense(units=128,activation="relu"))
model.add(Dense(units=128,activation="relu"))
model.add(Dense(units=64,activation="relu"))
model.add(Dense(units=1, activation="elu"))
```

```
opt=SGD(lr=0.001,momentum=0.9)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train,Y_train,validation_data=(X_test,Y_test),batch_size=128,epochs=10,verbose=1)
#model.save_weights("weights.h5")
#model.save("final.h5")
```

```
Epoch 6/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.6551 - acc: 0.6090 - val_los
s: 0.4959 - val_acc: 0.7988
Epoch 7/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.6826 - acc: 0.5666 - val_los
s: 0.5976 - val_acc: 0.7548
Epoch 8/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.6555 - acc: 0.6138 - val_los
s: 0.7102 - val_acc: 0.4795
Epoch 9/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.6380 - acc: 0.6460 - val_los
s: 0.4926 - val_acc: 0.8168
Epoch 10/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.6596 - acc: 0.6167 - val_los
s: 0.6174 - val acc: 0.6747
```

```
opt2=RMSprop(lr=0.001, rho=0.9)
model.compile(optimizer=opt2, loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train,Y_train,validation_data=(X_test,Y_test),batch_size=128,epochs=10,verbose=1)
#model.save_weights("weights.h5")
#model.save("final.h5")
```

```
Epoch 6/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
Epoch 7/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
Epoch 8/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
Epoch 9/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
Epoch 10/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
```

Activation funtion = sigmoid with all combinations of optimizers

```
import sys
from matplotlib import pyplot
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.optimizers import SGD


model = Sequential()
model.add(Conv2D(input_shape=(200,200,3),filters=64,kernel_size=(3,3),padding="same", activatio
n="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))


model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Flatten())

model.add(Dense(units=1024,activation="relu"))
model.add(Dense(units=1000,activation="relu"))
model.add(Dense(units=500,activation="relu"))
model.add(Dense(units=128,activation="relu"))
model.add(Dense(units=128,activation="relu"))
model.add(Dense(units=64,activation="relu"))
model.add(Dense(units=1, activation="elu"))
```

```
opt=SGD(lr=0.001,momentum=0.9)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train,Y_train,validation_data=(X_test,Y_test),batch_size=128,epochs=10,verbose=1)
#model.save_weights("weights.h5")
#model.save("final.h5")
```

```
Epoch 6/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.4177 - acc: 0.8076 - val_los
s: 0.3892 - val_acc: 0.8398
Epoch 7/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.3795 - acc: 0.8298 - val_los
s: 0.3765 - val_acc: 0.8529
Epoch 8/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.3798 - acc: 0.8315 - val_los
s: 0.3641 - val_acc: 0.8659
Epoch 9/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.3647 - acc: 0.8401 - val_los
s: 0.4272 - val_acc: 0.8278
Epoch 10/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.3362 - acc: 0.8631 - val_los
s: 0.2964 - val_acc: 0.8729
```

```
opt2=RMSprop(lr=0.001, rho=0.9)
model.compile(optimizer=opt2, loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train,Y_train,validation_data=(X_test,Y_test),batch_size=128,epochs=10,verbose=1)
#model.save_weights("weights.h5")
#model.save("final.h5")
```

```
Epoch 6/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
Epoch 7/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
Epoch 8/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
Epoch 9/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
Epoch 10/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
```

Activation funtion = hardsigmoid with all combinations of optimizers

```
import sys
from matplotlib import pyplot
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.optimizers import SGD


model = Sequential()
model.add(Conv2D(input_shape=(200,200,3),filters=64,kernel_size=(3,3),padding="same", activatio
n="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))


model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Flatten())

model.add(Dense(units=1024,activation="relu"))
model.add(Dense(units=1000,activation="relu"))
model.add(Dense(units=500,activation="relu"))
model.add(Dense(units=128,activation="relu"))
model.add(Dense(units=128,activation="relu"))
model.add(Dense(units=64,activation="relu"))
model.add(Dense(units=1, activation="hard_sigmoid"))
```

```
opt=SGD(lr=0.001,momentum=0.9)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train,Y_train,validation_data=(X_test,Y_test),batch_size=128,epochs=10,verbose=1)
#model.save_weights("weights.h5")
#model.save("final.h5")
```

```
Epoch 6/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.6923 - acc: 0.5573 - val_los
s: 0.6882 - val_acc: 0.8088
Epoch 7/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.6910 - acc: 0.5730 - val_los
s: 0.6855 - val_acc: 0.8098
Epoch 8/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.6882 - acc: 0.6383 - val_los
s: 0.6780 - val_acc: 0.7788
Epoch 9/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.6761 - acc: 0.6527 - val_los
s: 0.6557 - val_acc: 0.7327
Epoch 10/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.6869 - acc: 0.5541 - val_los
s: 0.6736 - val_acc: 0.7998
```

```
opt2=RMSprop(lr=0.001, rho=0.9)
model.compile(optimizer=opt2, loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train,Y_train,validation_data=(X_test,Y_test),batch_size=128,epochs=10,verbose=1)
#model.save_weights("weights.h5")
#model.save("final.h5")
```

```
Epoch 6/10
5604/5604 [==============================] - 38s 7ms/step - loss: 8.2841 - acc: 0.4804 - val_los
s: 12.7507 - val_acc: 0.2002
Epoch 7/10
5604/5604 [==============================] - 38s 7ms/step - loss: 8.2841 - acc: 0.4804 - val_los
s: 12.7507 - val_acc: 0.2002
Epoch 8/10
5604/5604 [==============================] - 38s 7ms/step - loss: 8.2841 - acc: 0.4804 - val_los
s: 12.7507 - val_acc: 0.2002
Epoch 9/10
5604/5604 [==============================] - 38s 7ms/step - loss: 8.2841 - acc: 0.4804 - val_los
s: 12.7507 - val_acc: 0.2002
Epoch 10/10
5604/5604 [==============================] - 38s 7ms/step - loss: 8.2841 - acc: 0.4804 - val_los
s: 12.7507 - val_acc: 0.2002
```

Activation funtion = selu with all combinations of optimizers

```
import sys
from matplotlib import pyplot
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.optimizers import SGD


model = Sequential()
model.add(Conv2D(input_shape=(200,200,3),filters=64,kernel_size=(3,3),padding="same", activatio
n="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))


model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Flatten())

model.add(Dense(units=1024,activation="relu"))
model.add(Dense(units=1000,activation="relu"))
model.add(Dense(units=500,activation="relu"))
model.add(Dense(units=128,activation="relu"))
model.add(Dense(units=128,activation="relu"))
model.add(Dense(units=64,activation="relu"))
model.add(Dense(units=1, activation="selu"))
```

```
opt=SGD(lr=0.001,momentum=0.9)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train,Y_train,validation_data=(X_test,Y_test),batch_size=128,epochs=10,verbose=1)
#model.save_weights("weights.h5")
#model.save("final.h5")
```

```
Epoch 6/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.6654 - acc: 0.6074 - val_los
s: 0.5506 - val_acc: 0.7798
Epoch 7/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.6441 - acc: 0.6445 - val_los
s: 0.7200 - val_acc: 0.5175
Epoch 8/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.6911 - acc: 0.5864 - val_los
s: 0.7538 - val_acc: 0.2002
Epoch 9/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.6946 - acc: 0.5159 - val_los
s: 0.6807 - val_acc: 0.7998
Epoch 10/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.6922 - acc: 0.5196 - val_los
s: 0.6631 - val_acc: 0.7998
```

```
opt2=RMSprop(lr=0.001, rho=0.9)
model.compile(optimizer=opt2, loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train,Y_train,validation_data=(X_test,Y_test),batch_size=128,epochs=10,verbose=1)
#model.save_weights("weights.h5")
#model.save("final.h5")
```

```
Epoch 6/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
Epoch 7/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
Epoch 8/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
Epoch 9/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
Epoch 10/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
```

Activation funtion = softsign with all combinations of optimizers

```
import sys
from matplotlib import pyplot
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.optimizers import RMSprop

model = Sequential()
model.add(Conv2D(input_shape=(200,200,3),filters=64,kernel_size=(3,3),padding="same", activatio
n="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))


model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Flatten())

model.add(Dense(units=1024,activation="relu"))
model.add(Dense(units=1000,activation="relu"))
model.add(Dense(units=500,activation="relu"))
model.add(Dense(units=128,activation="relu"))
model.add(Dense(units=128,activation="relu"))
model.add(Dense(units=64,activation="relu"))
model.add(Dense(units=1, activation="softsign"))
```

```
opt=SGD(lr=0.001,momentum=0.9)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train,Y_train,validation_data=(X_test,Y_test),batch_size=128,epochs=10,verbose=1)
#model.save_weights("weights.h5")
#model.save("final.h5")
```

```
Epoch 6/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.6880 - acc: 0.5293 - val_los
s: 0.6495 - val_acc: 0.7407
Epoch 7/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.6785 - acc: 0.5724 - val_los
s: 0.5597 - val_acc: 0.7998
Epoch 8/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.6482 - acc: 0.6494 - val_los
s: 0.5437 - val_acc: 0.8048
Epoch 9/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.7181 - acc: 0.5821 - val_los
s: 0.6659 - val_acc: 0.6957
Epoch 10/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.6831 - acc: 0.5494 - val_los
s: 0.7065 - val_acc: 0.4304
```

```
opt2=RMSprop(lr=0.001, rho=0.9)
model.compile(optimizer=opt2, loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train,Y_train,validation_data=(X_test,Y_test),batch_size=128,epochs=10,verbose=1)
#model.save_weights("weights.h5")
#model.save("final.h5")
```

```
Epoch 6/10
5604/5604 [==============================] - 39s 7ms/step - loss: 8.2841 - acc: 0.4804 - val_los
s: 12.7507 - val_acc: 0.2002
Epoch 7/10
5604/5604 [==============================] - 39s 7ms/step - loss: 8.2841 - acc: 0.4804 - val_los
s: 12.7507 - val_acc: 0.2002
Epoch 8/10
5604/5604 [==============================] - 39s 7ms/step - loss: 8.2841 - acc: 0.4804 - val_los
s: 12.7507 - val_acc: 0.2002
Epoch 9/10
5604/5604 [==============================] - 39s 7ms/step - loss: 8.2841 - acc: 0.4804 - val_los
s: 12.7507 - val_acc: 0.2002
Epoch 10/10
5604/5604 [==============================] - 39s 7ms/step - loss: 8.2841 - acc: 0.4804 - val_los
s: 12.7507 - val_acc: 0.2002
```

Activation funtion = softmax with all combinations of optimizers

```
import sys
from matplotlib import pyplot
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.optimizers import SGD

model = Sequential()
model.add(Conv2D(input_shape=(200,200,3),filters=64,kernel_size=(3,3),padding="same", activatio
n="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))


model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Flatten())

model.add(Dense(units=1024,activation="relu"))
model.add(Dense(units=1000,activation="relu"))
model.add(Dense(units=500,activation="relu"))
model.add(Dense(units=128,activation="relu"))
model.add(Dense(units=128,activation="relu"))
model.add(Dense(units=64,activation="relu"))
model.add(Dense(units=1, activation="softmax"))
```

```
opt=SGD(lr=0.001,momentum=0.9)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train,Y_train,validation_data=(X_test,Y_test),batch_size=128,epochs=10,verbose=1)
#model.save_weights("weights.h5")
#model.save("final.h5")
```

```
Epoch 6/10
5604/5604 [==============================] - 38s 7ms/step - loss: 8.2841 - acc: 0.4804 - val_los
s: 12.7507 - val_acc: 0.2002
Epoch 7/10
5604/5604 [==============================] - 38s 7ms/step - loss: 8.2841 - acc: 0.4804 - val_los
s: 12.7507 - val_acc: 0.2002
Epoch 8/10
5604/5604 [==============================] - 38s 7ms/step - loss: 8.2841 - acc: 0.4804 - val_los
s: 12.7507 - val_acc: 0.2002
Epoch 9/10
5604/5604 [==============================] - 38s 7ms/step - loss: 8.2841 - acc: 0.4804 - val_los
s: 12.7507 - val_acc: 0.2002
Epoch 10/10
5604/5604 [==============================] - 38s 7ms/step - loss: 8.2841 - acc: 0.4804 - val_los
s: 12.7507 - val_acc: 0.2002
```

```
opt2=RMSprop(lr=0.001, rho=0.9)
model.compile(optimizer=opt2, loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train,Y_train,validation_data=(X_test,Y_test),batch_size=128,epochs=10,verbose=1)
#model.save_weights("weights.h5")
#model.save("final.h5")
```

```
Epoch 6/10
5604/5604 [==============================] - 41s 7ms/step - loss: 8.2841 - acc: 0.4804 - val_los
s: 12.7507 - val_acc: 0.2002
Epoch 7/10
5604/5604 [==============================] - 41s 7ms/step - loss: 8.2841 - acc: 0.4804 - val_los
s: 12.7507 - val_acc: 0.2002
Epoch 8/10
5604/5604 [==============================] - 41s 7ms/step - loss: 8.2841 - acc: 0.4804 - val_los
s: 12.7507 - val_acc: 0.2002
Epoch 9/10
5604/5604 [==============================] - 41s 7ms/step - loss: 8.2841 - acc: 0.4804 - val_los
s: 12.7507 - val_acc: 0.2002
Epoch 10/10
5604/5604 [==============================] - 41s 7ms/step - loss: 8.2841 - acc: 0.4804 - val_los
s: 12.7507 - val_acc: 0.2002
```

Activation funtion = softplus with all combinations of optimizers

```
import sys
from matplotlib import pyplot
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.optimizers import SGD


model = Sequential()
model.add(Conv2D(input_shape=(200,200,3),filters=64,kernel_size=(3,3),padding="same", activatio
n="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))


model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Flatten())

model.add(Dense(units=1024,activation="relu"))
model.add(Dense(units=1000,activation="relu"))
model.add(Dense(units=500,activation="relu"))
model.add(Dense(units=128,activation="relu"))
model.add(Dense(units=128,activation="relu"))
model.add(Dense(units=64,activation="relu"))
model.add(Dense(units=1, activation="softplus"))
```

```
opt=SGD(lr=0.001,momentum=0.9)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train,Y_train,validation_data=(X_test,Y_test),batch_size=128,epochs=10,verbose=1)
#model.save_weights("weights.h5")
#model.save("final.h5")
```

```
Epoch 6/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.6663 - acc: 0.6128 - val_los
s: 0.5534 - val_acc: 0.8078
Epoch 7/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.6167 - acc: 0.6747 - val_los
s: 0.5125 - val_acc: 0.8308
Epoch 8/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.6108 - acc: 0.6854 - val_los
s: 0.4887 - val_acc: 0.8288
Epoch 9/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.6681 - acc: 0.5571 - val_los
s: 0.7320 - val_acc: 0.2402
Epoch 10/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.6789 - acc: 0.5708 - val_los
s: 0.5993 - val_acc: 0.7968
```

```
opt2=RMSprop(lr=0.001, rho=0.9)
model.compile(optimizer=opt2, loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train,Y_train,validation_data=(X_test,Y_test),batch_size=128,epochs=10,verbose=1)
#model.save_weights("weights.h5")
#model.save("final.h5")
```

```
Epoch 6/10
5604/5604 [==============================] - 47s 8ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
Epoch 7/10
5604/5604 [==============================] - 46s 8ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
Epoch 8/10
5604/5604 [==============================] - 47s 8ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
Epoch 9/10
5604/5604 [==============================] - 46s 8ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
Epoch 10/10
5604/5604 [==============================] - 46s 8ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
```

Activation funtion = relu with all combinations of optimizers

```
import sys
from matplotlib import pyplot
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.optimizers import SGD


model = Sequential()
model.add(Conv2D(input_shape=(200,200,3),filters=64,kernel_size=(3,3),padding="same", activatio
n="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))


model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Flatten())

model.add(Dense(units=1024,activation="relu"))
model.add(Dense(units=1000,activation="relu"))
model.add(Dense(units=500,activation="relu"))
model.add(Dense(units=128,activation="relu"))
model.add(Dense(units=128,activation="relu"))
model.add(Dense(units=64,activation="relu"))
model.add(Dense(units=1, activation="relu"))
```

```
opt=SGD(lr=0.001,momentum=0.9)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train,Y_train,validation_data=(X_test,Y_test),batch_size=128,epochs=10,verbose=1)
#model.save_weights("weights.h5")
#model.save("final.h5")
```

```
Epoch 6/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.7065 - acc: 0.5116 - val_los
s: 0.6725 - val_acc: 0.6807
Epoch 7/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.6623 - acc: 0.6429 - val_los
s: 0.5790 - val_acc: 0.8038
Epoch 8/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.6704 - acc: 0.6392 - val_los
s: 0.5253 - val_acc: 0.7998
Epoch 9/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.7003 - acc: 0.5064 - val_los
s: 0.6730 - val_acc: 0.7998
Epoch 10/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.6907 - acc: 0.5196 - val_los
s: 0.6677 - val_acc: 0.7998
```

```
opt2=RMSprop(lr=0.001, rho=0.9)
model.compile(optimizer=opt2, loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train,Y_train,validation_data=(X_test,Y_test),batch_size=128,epochs=10,verbose=1)
#model.save_weights("weights.h5")
#model.save("final.h5")
```

```
Epoch 6/10
5604/5604 [==============================] - 69s 12ms/step - loss: 7.7427 - acc: 0.5196 - val_lo
ss: 3.2268 - val_acc: 0.7998
Epoch 7/10
5604/5604 [==============================] - 70s 12ms/step - loss: 7.7427 - acc: 0.5196 - val_lo
ss: 3.2268 - val_acc: 0.7998
Epoch 8/10
5604/5604 [==============================] - 69s 12ms/step - loss: 7.7427 - acc: 0.5196 - val_lo
ss: 3.2268 - val_acc: 0.7998
Epoch 9/10
5604/5604 [==============================] - 69s 12ms/step - loss: 7.7427 - acc: 0.5196 - val_lo
ss: 3.2268 - val_acc: 0.7998
Epoch 10/10
5604/5604 [==============================] - 70s 12ms/step - loss: 7.7427 - acc: 0.5196 - val_lo
ss: 3.2268 - val_acc: 0.7998
```

Activation funtion = exponential with all combinations of optimizers

```
import sys
from matplotlib import pyplot
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.optimizers import SGD


model = Sequential()
model.add(Conv2D(input_shape=(200,200,3),filters=64,kernel_size=(3,3),padding="same", activatio
n="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))


model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Flatten())

model.add(Dense(units=1024,activation="relu"))
model.add(Dense(units=1000,activation="relu"))
model.add(Dense(units=500,activation="relu"))
model.add(Dense(units=128,activation="relu"))
model.add(Dense(units=128,activation="relu"))
model.add(Dense(units=64,activation="relu"))
model.add(Dense(units=1, activation="exponential"))
```

```
opt=SGD(lr=0.001,momentum=0.9)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train,Y_train,validation_data=(X_test,Y_test),batch_size=128,epochs=10,verbose=1)
#model.save_weights("weights.h5")
#model.save("final.h5")
```

```
Epoch 6/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.5418 - acc: 0.7261 - val_los
s: 0.4438 - val_acc: 0.8158
Epoch 7/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.5514 - acc: 0.7261 - val_los
s: 0.4544 - val_acc: 0.8438
Epoch 8/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.5145 - acc: 0.7559 - val_los
s: 0.4655 - val_acc: 0.8108
Epoch 9/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.6506 - acc: 0.5890 - val_los
s: 0.6604 - val_acc: 0.6997
Epoch 10/10
5604/5604 [==============================] - 38s 7ms/step - loss: 0.6696 - acc: 0.5876 - val_los
s: 0.6647 - val_acc: 0.6016
```

```
opt2=RMSprop(lr=0.001, rho=0.9)
model.compile(optimizer=opt2, loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train,Y_train,validation_data=(X_test,Y_test),batch_size=128,epochs=10,verbose=1)
#model.save_weights("weights.h5")
#model.save("final.h5")
```

```
Epoch 6/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
Epoch 7/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
Epoch 8/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
Epoch 9/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
Epoch 10/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val acc: 0.7998
```

Activation funtion = linear with all combinations of optimizers

```
import sys
from matplotlib import pyplot
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.optimizers import SGD


model = Sequential()
model.add(Conv2D(input_shape=(200,200,3),filters=64,kernel_size=(3,3),padding="same", activatio
n="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))


model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Flatten())

model.add(Dense(units=1024,activation="relu"))
model.add(Dense(units=1000,activation="relu"))
model.add(Dense(units=500,activation="relu"))
model.add(Dense(units=128,activation="relu"))
model.add(Dense(units=128,activation="relu"))
model.add(Dense(units=64,activation="relu"))
model.add(Dense(units=1, activation="linear"))
```

```
opt=SGD(lr=0.001,momentum=0.9)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train,Y_train,validation_data=(X_test,Y_test),batch_size=128,epochs=10,verbose=1)
#model.save_weights("weights.h5")
#model.save("final.h5")
```

```
Epoch 6/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
Epoch 7/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
Epoch 8/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
Epoch 9/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
Epoch 10/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
```

```
opt2=RMSprop(lr=0.001, rho=0.9)
model.compile(optimizer=opt2, loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train,Y_train,validation_data=(X_test,Y_test),batch_size=128,epochs=10,verbose=1)
#model.save_weights("weights.h5")
#model.save("final.h5")
```

```
Epoch 6/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
Epoch 7/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
Epoch 8/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
Epoch 9/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
Epoch 10/10
5604/5604 [==============================] - 38s 7ms/step - loss: 7.7427 - acc: 0.5196 - val_los
s: 3.2268 - val_acc: 0.7998
```

Final Analysis Details

Few combinations of optimizers and activation function working code and itś output is shown above. Here is the result of all the combinations shoen below:

ACTIVATION FUNCTION	tahn	elu	sigmoid	hard_sigmoid	selu	softsign	softmax	softplus	relu	exponential	linear
OPTIMIZER											
SGD	81.48	67.47	87.29	79.98	79.98	43.04	20.02	79.68	79.98	60.16	79.98
RMSprop	79.98	79.98	79.98	20.02	79.98	20.02	20.02	79.98	79.98	79.98	79.98
Adagrad	20.02	79.98	20.02	79.98	79.98	79.98	20.02	79.98	79.98	20.02	79.98
Adadelta	80.68	57.26	60.96	78.18	82.98	80.48	20.02	60.76	82.08	47.65	74.17
Adam	79.98	79.98	79.98	79.98	79.98	79.98	20.02	79.98	79.98	20.02	79.98
Adamax	20.02	79.98	20.02	20.02	79.98	79.98	20.02	79.98	79.98	79.98	79.98
Nadam	79.98	79.98	79.98	79.98	79.98	79.98	20.02	79.98	79.98	79.98	79.98

3.0.5 Module 5 : Creation of Chrome Extension

We tried to put our trained model on the web browser itself so that it works on all the websites. Thus being said we converted it in the form of an extension which when activated would work for all the websites you visit while it is activated. Our technology tries to cover the maximum websites possible. The extension works in Google Chrome as any other extension you may use or see. You will have to download and add it to the Google Chrome and give it the required permissions to scan the pages you visit while surfing on the internet. This extension consists of an already trained model and would not require training it again by the user. So once it is installed and given permissions, it is ready to be used.To create a Google chrome extension we have to first create a manifest file of json format so that the web browser gives us all the permissions to deploy our extension over every website. This file consists description regarding your extension like its name, version, icon, permissions required, browser actions, etc. This is a crucial file in creating the extension.Also we need to give content security to any external link used, as in our case "https://cdn.jsdelivr.net" which allows us to use tensorflow in .js files. Now to use our extension we created a content file which helps us retrieve an image from a webpage pass it through our model to classify it as nude or non-nude and then disable it accordingly. The procedure to this is all written in the content.js that is a JavaScript file. We also have to convert our CNN model to JavaScript format we do this using the content file as well. To make our extension work we deploy our original sequential trained model over the internet forum so that is easily available through the extension to whoever uses it.

Manifest.json

```
{
        "manifest_version" : 2,
        "name" : "Bluring OF Images",
        "version" : "1.0",
        "description" : "Blurs the the nude images at run time",
        "icons":{
                "128" : "icon128.jpeg",
                "48" : "icon48.jpeg",
                "16" : "icon16.jpeg"
        },
        "permissions":[
        "activeTab",
        "tabs",
        "https://*/"
    ],
        "browser_action":{
                "default_icon" : "icon16.jpeg",
                "default_popup" : "popup.html"
        },



        "content_security_policy": "script-src 'self' 'unsafe-eval' https://cdn.jsdelivr.net; object-src 'self'"


}
```

popup.html

As Manifest file has command for default popup it will first be redirected to popup.html file which has the link to acces the tensflow in .js file, which is further redirected to content.js file

```
<html>
    <head>
        <title>Awesome extension</title>
        <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs"></script>
        <script src="content1.js"></script>
    </head>
    <body>
        <p id="sources">There might be images here</p>
        <div>
    </div>
    </body>
</html>
```

Content.js file

Content.js file has the main coding done to get src's from web pages,resizing,batch processing,loading the model layers which is converted in .json format from .h5 format, and passing the image through model and finally prediction.
To load model, we need to use async function only else,it will give an error.As we are doing the modifications of the websites availabe on google chrome,we need to have permissions for the same.For this, we chrome.tabs.some syntax as be requirement,this can be easily seen in the code below.If the model finds the result to be nude,the src of the image is send to other .js file.As jquery is used in the file for bluring the image, we also need to run jquery.min.js file to operate it in other file as shown in code below.

```
async function loadmodel(image,src){
        const handler = '/models/model.json';
        let model;
        console.log(src);
        model=await tf.loadLayersModel(handler);
        const prediction = model.predict(image).dataSync()[0];
        console.log(prediction);
        if(prediction>=0.4){
                chrome.tabs.query({
                active: true,
                currentWindow: true
                }, function (tabs) {
                chrome.tabs.executeScript(tabs[0].id,{file:"jquery.min.js"},function(){
                chrome.tabs.executeScript(tabs[0].id,{file:"content.js"},function(){
                chrome.tabs.sendMessage(tabs[0].id, {scriptOptions: {param1:src}}, function(){
                                        });
                                });
                        });
                });
        }
        };
```

```
22
23  async function loadImage(src) {
24      var img = new Image();
25      img.src = src;
26          img.setAttribute('crossOrigin','anonymous');
27          img.onload = function(){
28          const a=tf.browser.fromPixels(img);
29          //console.log("frompixels",a);
30          const processedImage = loadAndProcessImage(a,src);
31          loadmodel(processedImage,src);
32          }
33  };
34
35  function resizeImage(image) {
36    return tf.image.resizeBilinear(image, [200, 200]);
37  };
38
39  function batchImage(image) {
40    const batchedImage = image.expandDims(0);
41    return batchedImage.toFloat().div(tf.scalar(127)).sub(tf.scalar(1));
42  };
43
```

```
43
44  function loadAndProcessImage(image,src) {
45    const resizedImage = resizeImage(image);
46    //console.log("resizes",resizedImage);
47    const batchedImage = batchImage(resizedImage);
48    //console.log("batches",batchedImage);
49    return(batchedImage);
50  };
51
52  var callback = function (results) {
53
54      document.body.innerHTML = '';
55      for (var i in results[0]) {
56          img = document.createElement('img');
57          img.src = results[0][i];
58          z=img.src;
59          //console.log(img);
60          const image=loadImage(z);
61  } };
62
63
64
65  chrome.tabs.query({
66      active: true,
67      currentWindow: true
68  }, function (tabs) {
69      chrome.tabs.executeScript(tabs[0].id, {
70          code: 'Array.prototype.map.call(document.images, function (i) { return i.src; });'
71      }, callback);
72  });
```

content1.js file

Here, we are actually blurring the images,the main content.js file will sen the src of the image which we want to blur.Using css in jquery, we blur the image using blur filter by 10px.

```
1  function dosomething(a){
2          console.log(a);
3          $("img[src='"+a+"']").css( 'filter', 'blur(10px)');
4  }
5
6
7  chrome.runtime.onMessage.addListener(function(message, sender, sendResponse) {
8      var scriptOptions = message.scriptOptions;
9      //console.log('param1', scriptOptions.param1);
10     dosomething(scriptOptions.param1);
11 });
```

Chapter 4

Testing

As we have created extension,we need to first test it before deploying it.We go on"chrome://extensions" and load our chrome extension codes on it for testing.First on the developers mode and then click on Load Package and upload the code as shown below.

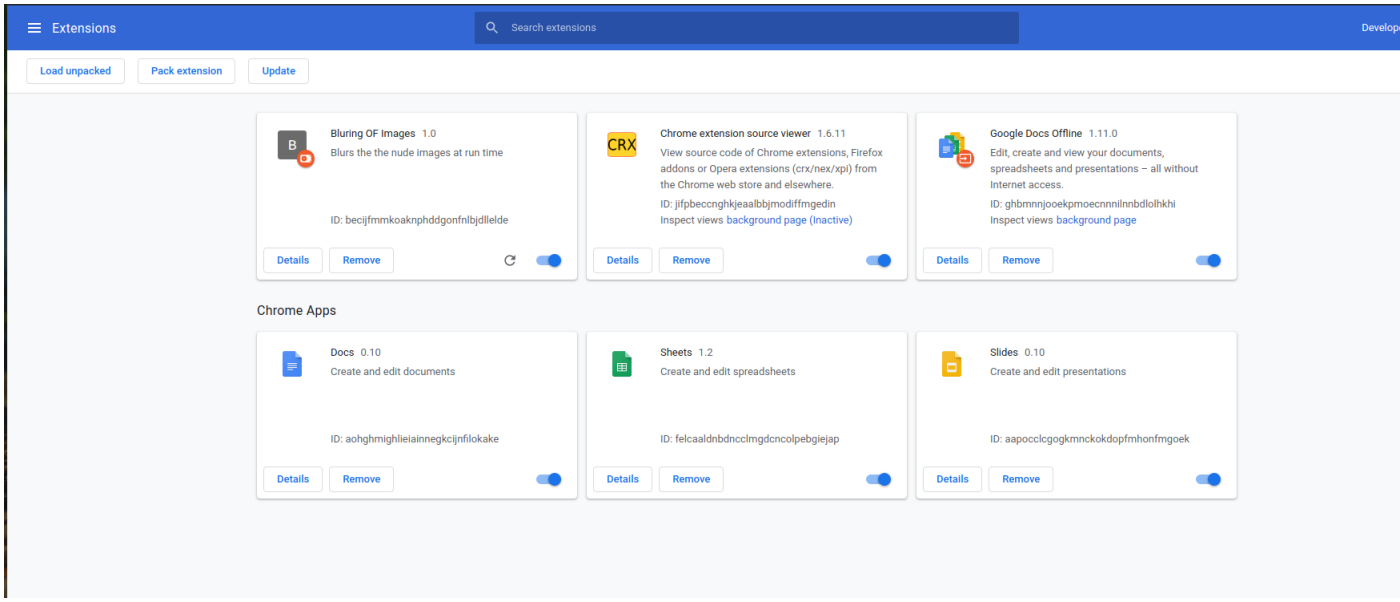

Figure 4.1: Uploading the extension code

4.0.1 Design of test cases

We have created a small html pages having 4 images ,2 nude and 2 nonnude images to test whether our extension code is working correctly or not. The code demo html page goes as follows:

```
<html>
<head>
        <img src="http://i.imgur.com/0cN7loZ.jpg" height="200" width="200">
        <img src="https://cdn.pornpics.com/pics1/2018-07-16/539074_06big.jpg" height="200" width="200">
        <img src="https://i.imgur.com/lErhNQs.jpg" height="200" width="200">
        <img src="https://i.imgur.com/rv3IZiN.jpg" height="200" width="200">
</head>
</html>
```

Onceś the scrś of the image is received,following details of the images are shown below:

```
t {kept: false, isDisposedInternal: false, shape: Array(3), dtype: "int32", size: 4320000, …}
  dataId: {}
  dtype: "int32"
  id: 10
  isDisposedInternal: false
  kept: false
  rankType: "3"
  scopeId: 15
  shape: (3) [1600, 900, 3]
  size: 4320000
  strides: (2) [2700, 3]
  isDisposed: (...)
  rank: (...)
  __proto__ : Object
```

Figure 4.2: Image Data

After resizeing the shape of the image is changes to (200,200,3) and datatype is changed to 'float32'

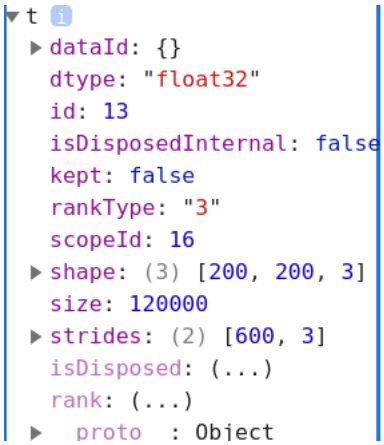

Figure 4.3: Image Data after resizing

Next comes batching,were shape of the image changes to 1*4 dimensions (1,200,200,3) which is one of the requirements of passing the images from model.

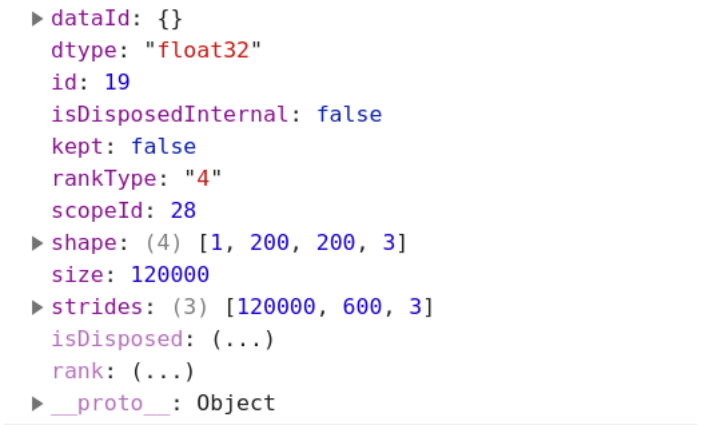

Figure 4.4: Image Data after batching

Prediction values

Figure 4.5: Prediction Value

Final demo output with images getting blurred

Example 1:

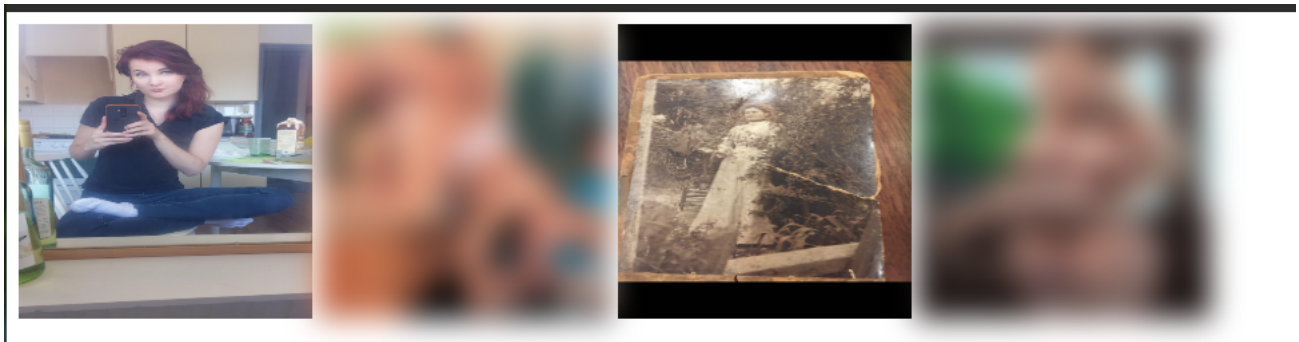

Figure 4.6: Demo-1

Example 2:

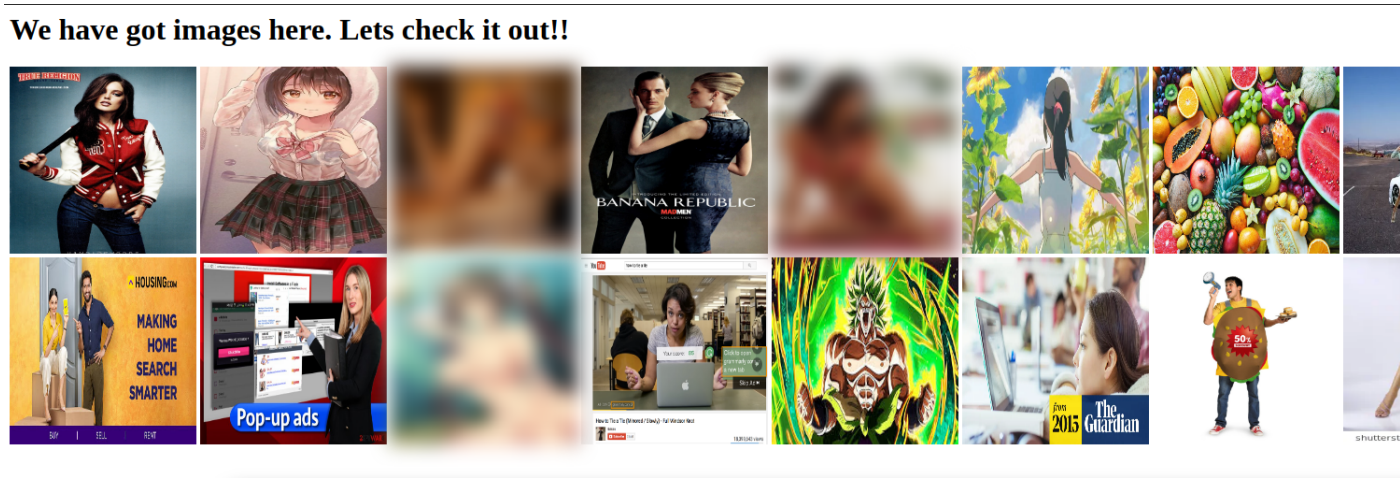

Figure 4.7: Demo-2

4.0.2 Testing

Here are few of the examples of the images getting blurred on the website available on Google Chrome

Example of website 1:

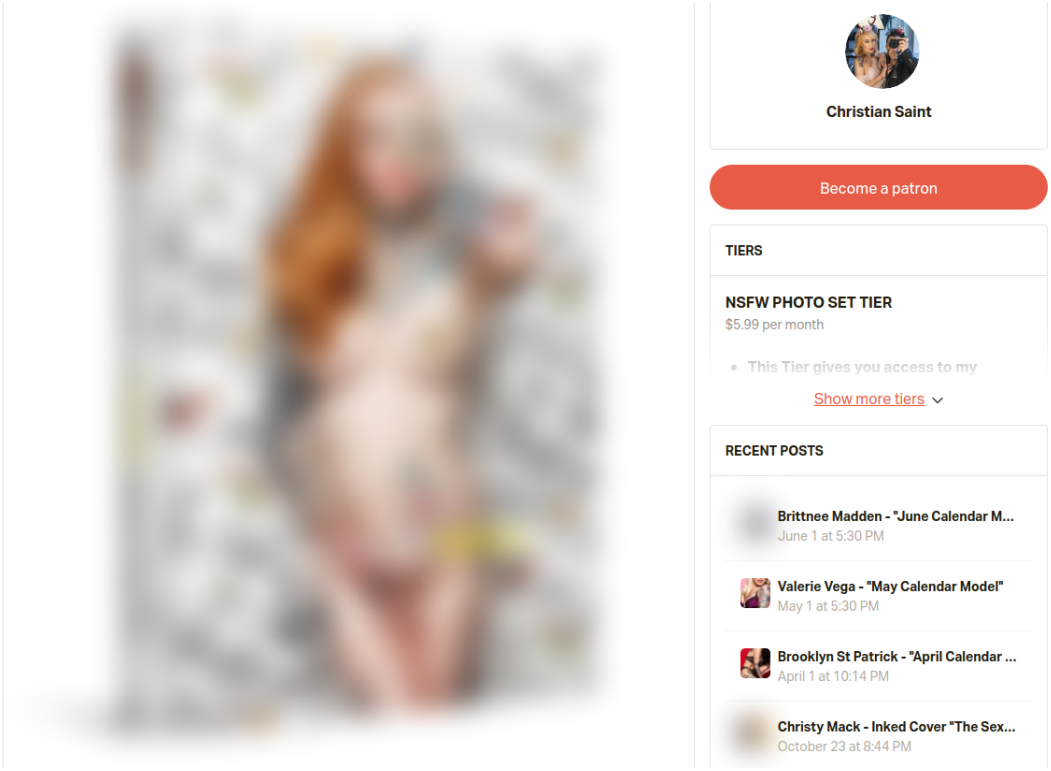

Figure 4.8: Example-1

Example of website 2:

This website has 100's of images in column here are few examples:

Figure 4.9: Example-1

Figure 4.10: Example-2

Figure 4.11: Example-3

Chapter 5

Results and Analysis

5.0.1 Graphical Representation

Bar Graph Below represents the time taken to blur no. of images on website

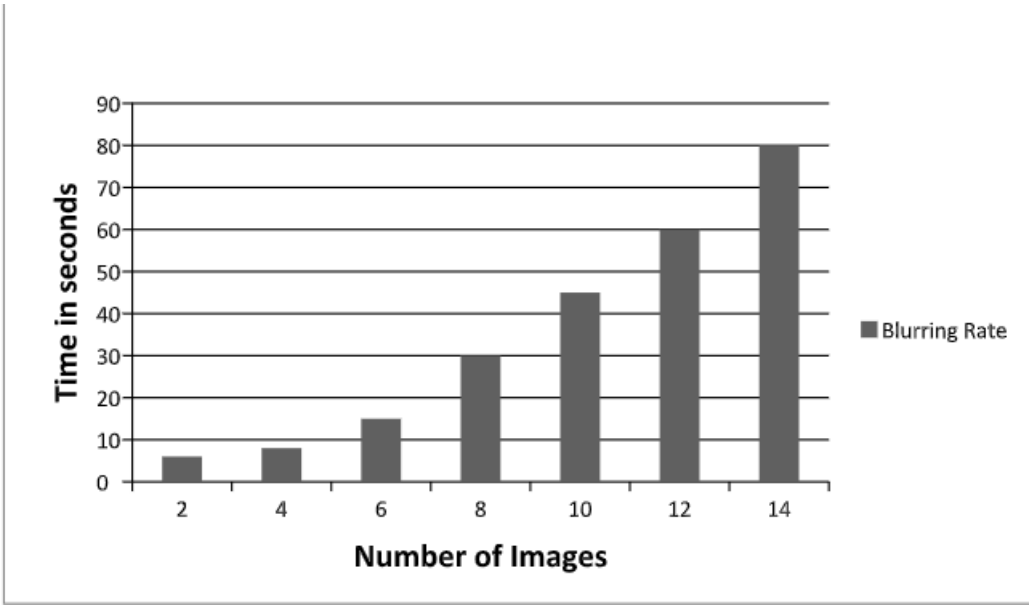

Figure 5.1: Bar Graph

Sr. No.	Number of Images	Time taken to blur the images(in seconds)
1	02	06
2	04	08
3	06	15
4	08	30
5	10	45
6	12	60
7	14	80

Figure 5.2: Details

5.0.2 Analysis of Result

We can analyse from the bar graph above that as the number of images on a web page increase, the time taken by the extension and model to process and blur those images also increase. This time forms an upward bar-graph. The images as and when increase needs to be filtered through the CNN as nude and non-nude and then blur these images if it crosses the minimum threshold. The other factor that contributes to this delay is the loading of CNN model. Each image loads the CNN model separately to be labelled as nude or non-nude this increases the time taken to blur these images as it puts an immense amount of time to load the model again and again for every image it encounters on the page. To overcome this problem and to ensure all the images are blurred almost in the same time frame of 2-5 seconds we need to load the CNN model only once and pass all the images through it at a single point of time.
From all the pages we tested our extension on we can analyse that the extension works splendidly and accurately. It correctly filters the images as nude and non-nude. It also blurs the nude images completely and leaves the non-nude images as it is. The extension also works on all the web pages on the internet and blurs the images labelled as nude. The extension matches all the bars of accuracy and consistency. The only lagging factor is the amount of time it takes to blur all those images.

Chapter 6

Future Scope

Our scope is to blur all the extreme images and deactivating their links. This means that it will scan the entire web page and find all the extreme images present on that page. Then we will blur those images before they are loaded, so that the children would not be able to see them. This will be done in a 2 step process first it will identify the image as nude or non-nude image. The nude image is blurred and non-nude image is displayed as it is. Our future scope is to reduce the amount of time it takes to blur the nude images. For this we will change our code and try to load the CNN model just once when the page is loaded and parse all the images displayed through it all at once. This will definitely reduce the time taken to blur these images to a minimum. We will also try to disable the redirecting links present on an image, thus ensuring double protection from the extremities of the internet.

Bibliography

[1] 2017 9th International Conference on Knowledge and Systems Engineering(KSE) Advertisement Image Classification Using Convolutional Neural Network

[2] https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/

[3] https://medium.com/@alkeshab/face-detection-using-opencv-in-google-colaboratory-a7529a2bb921

[4] https://www.analyticsvidya.com/blog/2017/06/architecture-of-convolution-neural-networks-simplified-demystified/

[5] https://neurohive.io/en/popular-networks/vgg16/

[6] https://www.pyimagesearch.com/2018/09/24/open-cv-face-recognition/

[7] https://www.willberger.org/cascade-haar-explained/

[8] https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d

[9] https://keras.io

[10] https://www.tensorflow.org/js

[11] https://thekevinscott.com/image-classification-with-javascript/

Acknowledgement

We have great pleasure in presenting the report on **"Real Time Bare Skinned Image Classification Using CNN.** We take this opportunity to express our sincere thanks towards our guide **Prof.Jaya Gupta** & Co-Guide **Prof.Amol Kalugade** Department of Computers, APSIT thane for providing the technical guidelines and suggestions regarding line of work. We would like to express our gratitude towards his constant encouragement, support and guidance through the development of project.

We thank **Prof.Sachin Malve** Head of Department,Computers, APSIT for his encouragement during progress meeting and providing guidelines to write this report.

We thank **Prof.Amol Kalugade** BE project co-ordinator, Department of Computers, APSIT for being encouraging throughout the course and for guidance.

We also thank the entire staff of APSIT for their invaluable help rendered during the course of this work. We wish to express our deep gratitude towards all our colleagues of APSIT for their encouragement.

Student Name1:Pratik Jain
Student ID1:16102060

Student Name2:Tina Shah
Student ID2:16102019

Student Name3:Kshitija Shah
Student ID3:16102027