**DESIGN PHASE**

**SYSTEM DESIGN**

**Introduction**

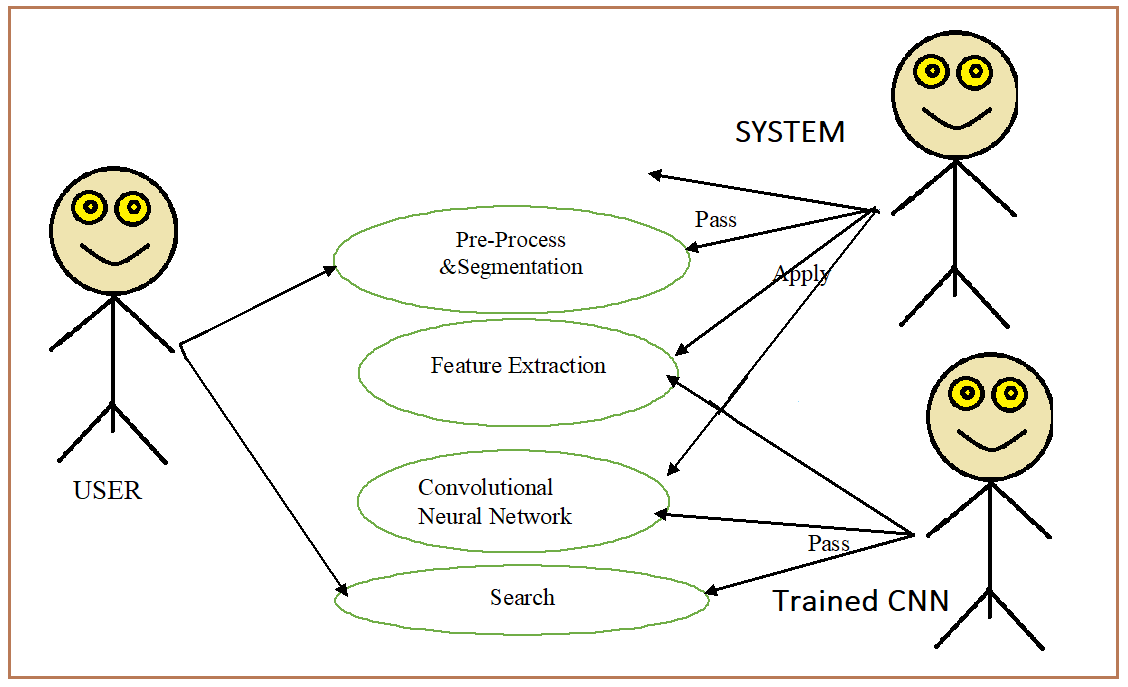
Project design is an early phase of the project where a project's key features, structure, criteria for success, and major deliverables are all planned out. The point is to develop one or more designs which can be used to achieve the desired project goals.

**Architectural Design**

The architectural diagram is a basic view of any system application to be developed. It is a foundation layout of the project works. With regard to that, the architectural design of the project of Car Brand Classification is depicted follows:

**Use case Diagram**

The use case view models functionality of the system as perceived by outside uses. A use case is a coherent unit of functionality expressed as a transaction among actors and the system.



#### **Sequence Diagram**

A sequence diagram is a graphical view of a scenario that shows object interaction in a time- based sequence what happens first, what happens next. Sequence diagram establish the role of objects and helps provide essential information to determine class responsibilities and interfaces. This type of diagram is best used during early analysis phase in design because they are simple and easy to comprehend. Sequence diagram are normally associated with use cases.

S: System

U: User

Select Test Image ()

Load Image ()

Invert Image ()

Button Clicking ()

Transfer Learning ()

Predict Image ()

Display Result ()

S: System

U: User

Get Train Image ()

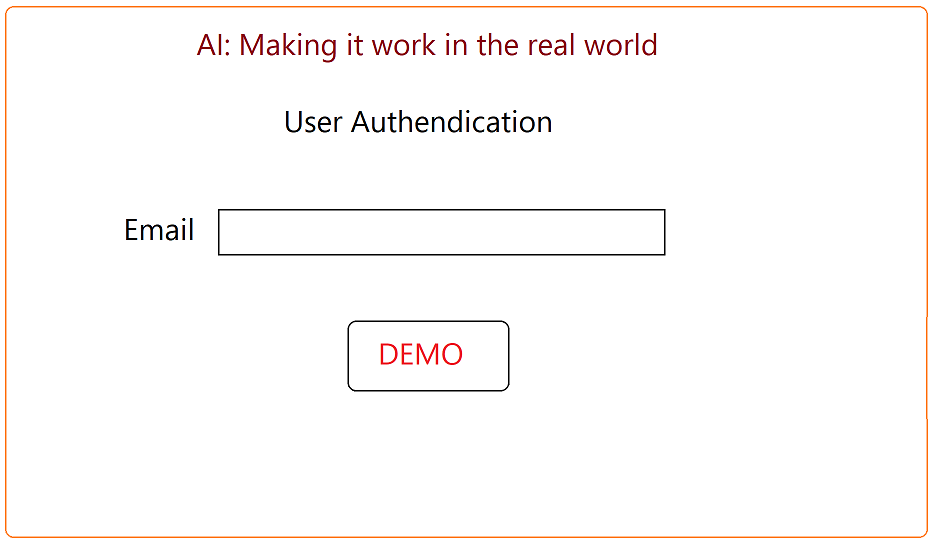
Process

Save Image ()

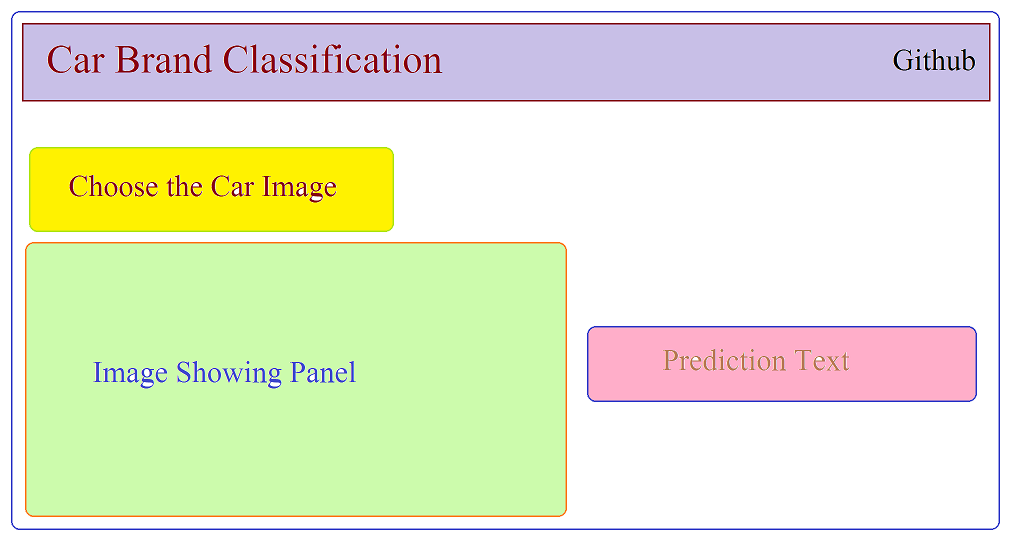
**PSEUDO FORM DESIGN**

Pseudo form designing means deciding the contents and layout of forms for the purpose of collecting and processing the required information economically and efficiently.

**User Authentication Page**



**User Interface:**



**Table Design:**

In the table design I’m going to show you which are algorithms are going to do use in this project in the table format.

State Of Art Algorithms Available Models:

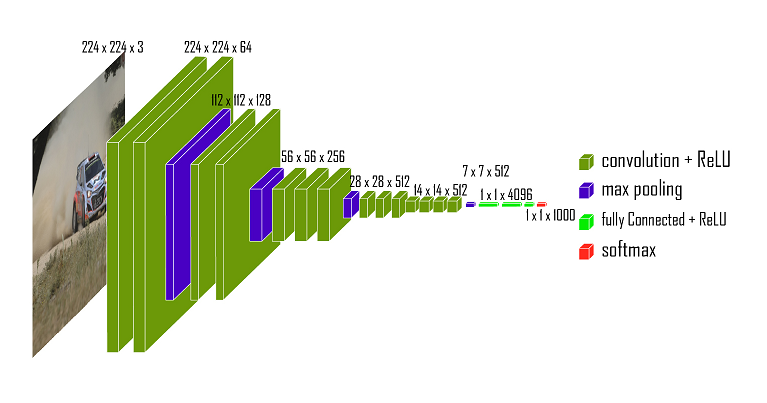
| **Model** | **Size** | **Top-1 Accuracy** | **Top-5 Accuracy** | **Parameters** | **Depth** |
| --- | --- | --- | --- | --- | --- |
| [Xception](https://keras.io/api/applications/xception) | 88 MB | 0.790 | 0.945 | 22,910,480 | 126 |
| [VGG16](https://keras.io/api/applications/vgg/#vgg16-function) | 528 MB | 0.713 | 0.901 | 138,357,544 | 23 |
| [VGG19](https://keras.io/api/applications/vgg/#vgg19-function) | 549 MB | 0.713 | 0.900 | 143,667,240 | 26 |
| [ResNet50](https://keras.io/api/applications/resnet/#resnet50-function) | 98 MB | 0.749 | 0.921 | 25,636,712 | - |
| [ResNet101](https://keras.io/api/applications/resnet/#resnet101-function) | 171 MB | 0.764 | 0.928 | 44,707,176 | - |
| [ResNet152](https://keras.io/api/applications/resnet/#resnet152-function) | 232 MB | 0.766 | 0.931 | 60,419,944 | - |
| [ResNet50V2](https://keras.io/api/applications/resnet/#resnet50v2-function) | 98 MB | 0.760 | 0.930 | 25,613,800 | - |
| [ResNet101V2](https://keras.io/api/applications/resnet/#resnet101v2-function) | 171 MB | 0.772 | 0.938 | 44,675,560 | - |
| [ResNet152V2](https://keras.io/api/applications/resnet/#resnet152v2-function) | 232 MB | 0.780 | 0.942 | 60,380,648 | - |
| [InceptionV3](https://keras.io/api/applications/inceptionv3) | 92 MB | 0.779 | 0.937 | 23,851,784 | 159 |
| [InceptionResNetV2](https://keras.io/api/applications/inceptionresnetv2) | 215 MB | 0.803 | 0.953 | 55,873,736 | 572 |
| [MobileNet](https://keras.io/api/applications/mobilenet) | 16 MB | 0.704 | 0.895 | 4,253,864 | 88 |
| [MobileNetV2](https://keras.io/api/applications/mobilenet/#mobilenetv2-function) | 14 MB | 0.713 | 0.901 | 3,538,984 | 88 |
| [DenseNet121](https://keras.io/api/applications/densenet/#densenet121-function) | 33 MB | 0.750 | 0.923 | 8,062,504 | 121 |
| [DenseNet169](https://keras.io/api/applications/densenet/#densenet169-function) | 57 MB | 0.762 | 0.932 | 14,307,880 | 169 |
| [DenseNet201](https://keras.io/api/applications/densenet/#densenet201-function) | 80 MB | 0.773 | 0.936 | 20,242,984 | 201 |
| [NASNetMobile](https://keras.io/api/applications/nasnet/#nasnetmobile-function) | 23 MB | 0.744 | 0.919 | 5,326,716 | - |
| [NASNetLarge](https://keras.io/api/applications/nasnet/#nasnetlarge-function) | 343 MB | 0.825 | 0.960 | 88,949,818 | - |
| [EfficientNetB0](https://keras.io/api/applications/efficientnet/#efficientnetb0-function) | 29 MB | - | - | 5,330,571 | - |
| [EfficientNetB1](https://keras.io/api/applications/efficientnet/#efficientnetb1-function) | 31 MB | - | - | 7,856,239 | - |
| [EfficientNetB2](https://keras.io/api/applications/efficientnet/#efficientnetb2-function) | 36 MB | - | - | 9,177,569 | - |
| [EfficientNetB3](https://keras.io/api/applications/efficientnet/#efficientnetb3-function) | 48 MB | - | - | 12,320,535 | - |
| [EfficientNetB4](https://keras.io/api/applications/efficientnet/#efficientnetb4-function) | 75 MB | - | - | 19,466,823 | - |
| [EfficientNetB5](https://keras.io/api/applications/efficientnet/#efficientnetb5-function) | 118 MB | - | - | 30,562,527 | - |
| [EfficientNetB6](https://keras.io/api/applications/efficientnet/#efficientnetb6-function) | 166 MB | - | - | 43,265,143 | - |
| [EfficientNetB7](https://keras.io/api/applications/efficientnet/#efficientnetb7-function) | 256 MB | - | - | 66,658,687 | - |

**Algorithms used in this Project:**

| **Model** | **Size** | **Top-1 Accuracy** | **Top-5 Accuracy** | **Parameters** | **Depth** |
| --- | --- | --- | --- | --- | --- |
| [VGG16](https://keras.io/api/applications/vgg/#vgg16-function) | 528 MB | 0.713 | 0.901 | 138,357,544 | 23 |
| [VGG19](https://keras.io/api/applications/vgg/#vgg19-function) | 549 MB | 0.713 | 0.900 | 143,667,240 | 26 |
| [ResNet50](https://keras.io/api/applications/resnet/#resnet50-function) | 98 MB | 0.749 | 0.921 | 25,636,712 | - |
| [ResNet101](https://keras.io/api/applications/resnet/#resnet101-function) | 171 MB | 0.764 | 0.928 | 44,707,176 | - |
| [ResNet152](https://keras.io/api/applications/resnet/#resnet152-function) | 232 MB | 0.766 | 0.931 | 60,419,944 | - |
| [ResNet50V2](https://keras.io/api/applications/resnet/#resnet50v2-function) | 98 MB | 0.760 | 0.930 | 25,613,800 | - |

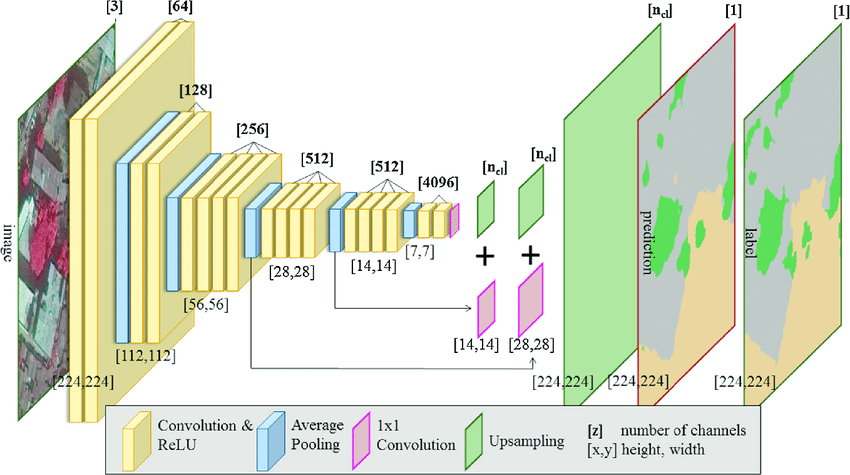
# **VGG-16 | CNN model**

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is an annual computer vision competition. Each year, teams compete on two tasks. The first is to detect objects within an image coming from 200 classes, which is called object localization. The second is to classify images, each labeled with one of 1000 categories, which is called image classification. VGG 16 was proposed by Karen Simonyan and Andrew Zisserman of the Visual Geometry Group Lab of Oxford University in 2014 in the paper “VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION”. This model won the 1stand 2nd place on the above categories in 2014 ILSVRC challenge.



**VGG-19 | CNN model**

**VGG**-**19** is a convolutional neural network that is **19** layers deep. You can load a pretrained version of the network trained on more than a million images from the ImageNet database. The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals.



**Frameworks:**

**Tensorflow:**

**TensorFlow** is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

**Keras:**

Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible.

**Project Description**

The system after careful analysis has been identified that there are three main modules and each modules comprises of several sub modules. The main modules are as follows,

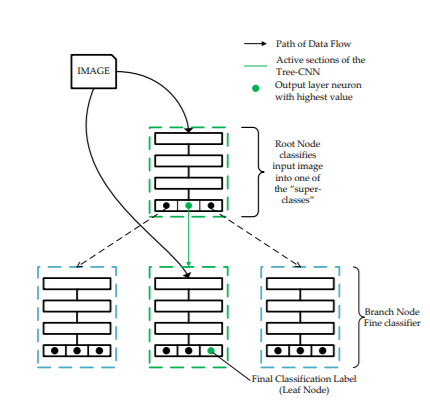
* Image Pre-Processing
* Classifying Car Images using CNN
* Performing Transfer Learning
* Get the Prediction
* Deploying as a Web Application in Heroku

**CNN:**

In deep learning, a convolutional neural network is a class of deep neural networks, most commonly applied to analyzing visual imagery. They are also known as shift invariant or space invariant artificial neural networks, based on their shared-weights architecture and translation invariance characteristics

**CNN Architecture for Image Classification:**

Convolutional Neural Networks (CNNs) leverage spatial information, and they are therefore well suited for classifying images. These networks use an ad hoc architecture inspired by biological data taken from physiological experiments performed on the visual cortex. Our vision is based on multiple cortex levels, each one recognizing more and more structured information. First, we see single pixels, then from that we recognize simple geometric forms, and more sophisticated elements such as objects, faces, human bodies, animals, and so on

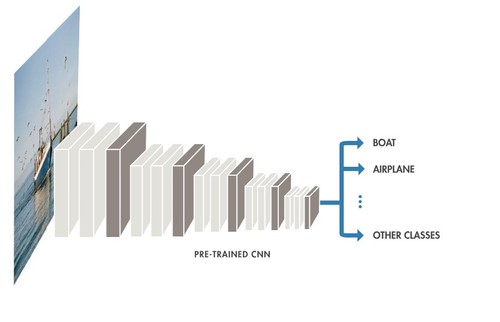


**Transfer Learning:**

Transfer learning is the idea of overcoming the isolated learning paradigm and

Utilizing knowledge acquired for one task to solve related ones.

**Architecture of Transfer Learning:**



**Heroku:**

Heroku is a cloud platform as a service supporting several programming

languages. One of the first cloud platforms, Heroku has been in development since

June 2007, when it supported only the Ruby programming language, but now

supports Java, Node.js, Scala, Clojure, Python, PHP, and Go.

**Pseudocode for Image Classification in CNN:**

I = Input Image, node = Root Node of the Tree

procedure ClassPredict(I, node)

count = # of children of node

if count = 0 then

label = class label of the node

return label

else

nextNode = EvaluateNode(I, node)

I returns the address of the child node of highest output neuron

return ClassPredict(I, nextNode)

end if

end procedure

**Generate CNN Tree for Classifying Images:**

L = Likelihood Matrix

maxChildren = max. number of children per branch node

RootNode = Root Node of the Tree-CNN

procedure GrowTree(L, Node)

S = GenenerateS(L, Node, maxChildren)

while S is not Empty do

I Get attributes of the first object

[label, value, node] = Get Attributes(S[1])

if value[1] − value[2] > α then

I The new class has a strong preference for n1

I Adds label to node[1]

RootNode = AddClasstoNode(RootNode, label, node[1])

else

if value[2] − value[3] > β then

I The new class has similar strong preference n1 and n2

Merge = Check f orMerge(Node, node[1], node[2])

I Merge is True only if node[2] is a leaf node, and,

I the # of children of node[1] less than maxChildren − 1

if Merge then

I Merge node[2] into node[1]

RootNode = MergeNode(RootNode, node[1], node[2])

RootNode = AddClasstoNode(RootNode, label, node[1])

else

I Add new class to the smaller output node

sNode = Node with lesser children (node[1], node[2])

RootNode = AddClasstoNode(RootNode, label, sNode)

end if

else

I Add new class as a new Leaf node to Root Node

RootNode = AddNewNode(RootNode, label)

end if

end if

I Remove the columns of the added class from L

I Remove the rows of “full” nodes from L

I Regenerate S

S = GenenerateS(L, Node, maxChildren)

end while

end procedure