## horizontal line



Damage Detection AI

URL: <https://damage-detection-model.herokuapp.com>

09.06.2018

**─**

Y Abhilash Reddy

Insofe (International School of Engineering)

Hyderabad

Telangana

# CONTENTS:

Overview

Goals

Tools and Packages Used

Milestones

1. Deep Learning Model Convolutional Network
2. Flask
3. Heroku
4. Final Summary

# 

# 

# 

# 

# 

# 

# 

# Overview

The major set back in manufacturing line is that segregating the parts which are damaged and which are not. Due to which the department faces shortage of the parts or increase in capital.

So, in order to segregate the parts we will be using Artificial Intelligence to detect whether the spare part is damaged or not.

# Goals

1. Build an AI model Which segregates the images based on the classes given.
2. Integrate the AI model with Server and deploying it live so that anyone can use that app directly and test the class it belongs to.

# Tools Used and Packages Used

**Deep Learning Model:** For model building and the architecture used to design is Convolutional 2D.

**Flask:** Making the app which can be hosted on to server for deploying them live.

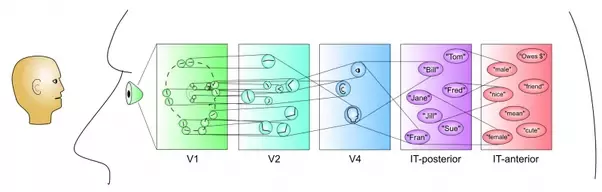
**Heroku:** Web server to deploy the model live.

# Milestones

## **Deep learning Model Convolutional networks**

## **What inspired Convolutional Networks?**

CNNs are biologically-inspired models inspired by research by D. H. Hubel and T. N. Wiesel. They proposed an explanation for the way in which mammals visually perceive the world around them using a layered architecture of neurons in the brain, and this in turn inspired engineers to attempt to develop similar pattern recognition mechanisms in computer vision.



In their hypothesis, within the visual cortex, complex functional responses generated by "complex cells" are constructed from more simplistic responses from "simple cells'.

For instances, simple cells would respond to oriented edges etc, while complex cells will also respond to oriented edges but with a degree of spatial invariance.

Receptive fields exist for cells, where a cell responds to a summation of inputs from other local cells.

The architecture of deep convolutional neural networks was inspired by the ideas mentioned above

* local connections
* layering
* spatial invariance (shifting the input signal results in an equally shifted output signal. , most of us are able to recognize specific faces under a variety of conditions because we learn abstraction These abstractions are thus invariant to size, contrast, rotation, orientation

However, it remains to be seen if these computational mechanisms of convolutional neural networks are similar to the computation mechanisms occurring in the primate visual system

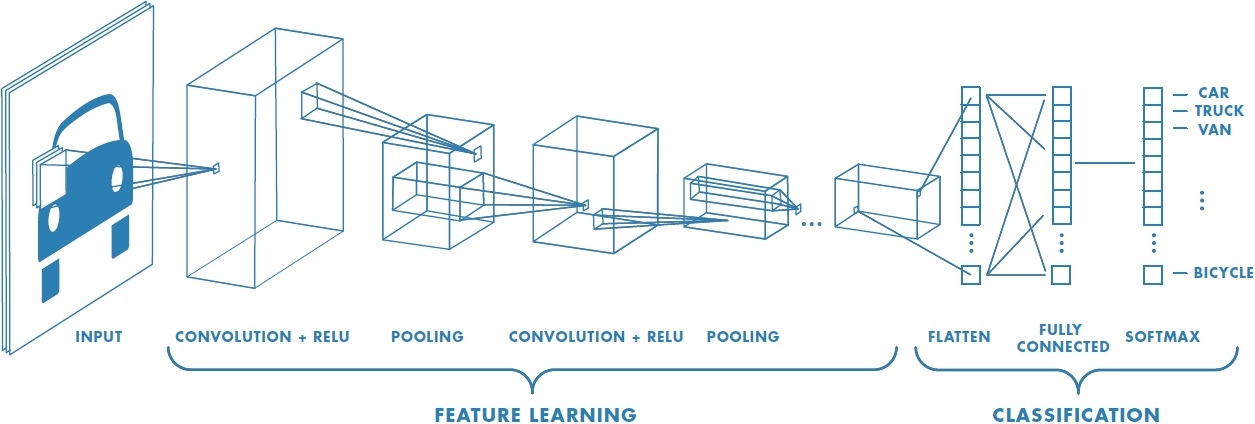
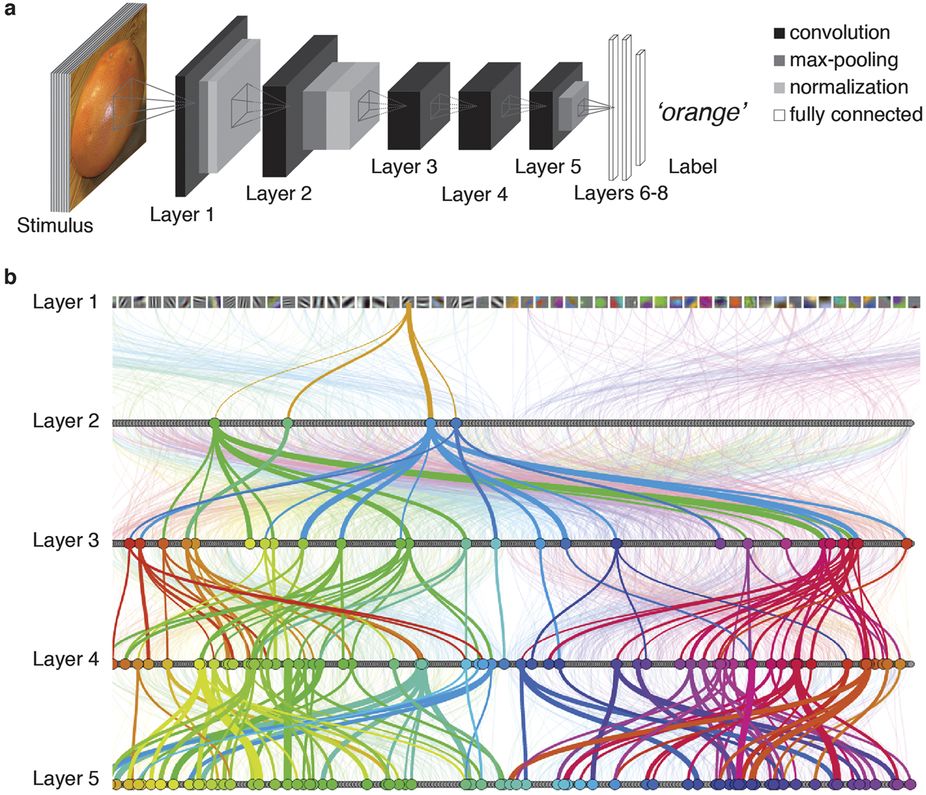
* convolution operation
* shared weights
* pooling/subsampling

## 

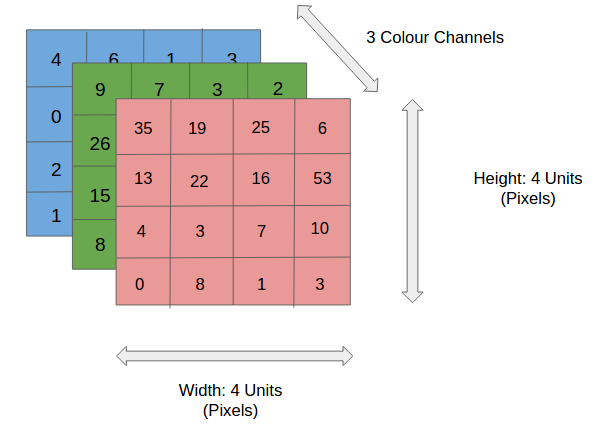
## 

## 

## **How does it work?**

****

### **Step 1 - Prepare a dataset of images**

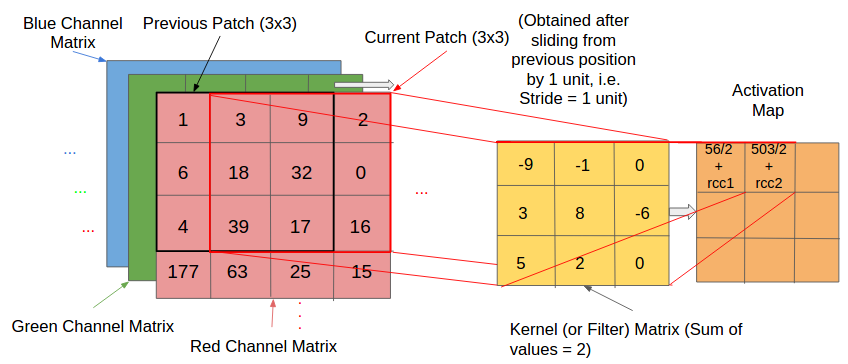
****

* Every image is a matrix of pixel values.
* The range of values that can be encoded in each pixel depends upon its bit size.
* Most commonly, we have 8 bit or 1 Byte-sized pixels. Thus the possible range of values a single pixel can represent is [0, 255].
* However, with coloured images, particularly RGB (Red, Green, Blue)-based images, the presence of separate colour channels (3 in the case of RGB images) introduces an additional ‘depth’ field to the data, making the input 3-dimensional.
* Hence, for a given RGB image of size, say 150×150 (Width x Height) pixels, we’ll have 3 matrices associated with each image, one for each of the colour channels.
* Thus the image in its entirety, constitutes a 3-dimensional structure called the Input Volume (150x150x3).

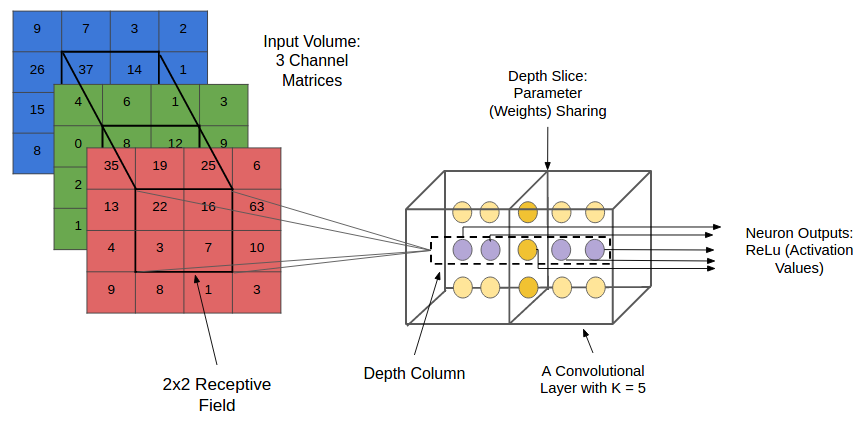
Great training datasets are damaged Spare Parts which has to be classified which are given.

### **Step 2 - Convolution**

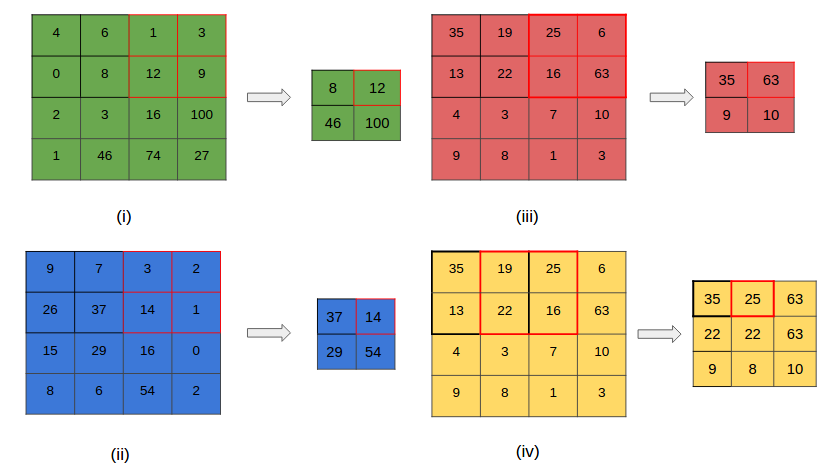
****

****

* A convolution is an orderly procedure where two sources of information are intertwined.
* A kernel (also called a filter) is a smaller-sized matrix in comparison to the input dimensions of the image, that consists of real valued entries.
* Kernels are then convolved with the input volume to obtain so-called ‘activation maps’ (also called feature maps).
* Activation maps indicate ‘activated’ regions, i.e. regions where features specific to the kernel have been detected in the input.
* The real values of the kernel matrix change with each learning iteration over the training set, indicating that the network is learning to identify which regions are of significance for extracting features from the data.
* We compute the dot product between the kernel and the input matrix. -The convolved value obtained by summing the resultant terms from the dot product forms a single entry in the activation matrix.
* The patch selection is then slided (towards the right, or downwards when the boundary of the matrix is reached) by a certain amount called the ‘stride’ value, and the process is repeated till the entire input image has been processed. - The process is carried out for all colour channels.
* instead of connecting each neuron to all possible pixels, we specify a 2 dimensional region called the ‘receptive field[14]’ (say of size 2×2 units) extending to the entire depth of the input (2x2x3 for a 3 colour channel input), within which the encompassed pixels are fully connected to the neural network input layer. It’s over these small regions that the network layer cross-sections (each consisting of several neurons (called ‘depth columns’)) operate and produce the activation map. (reduces computational complexity)



### **Step 3 - Pooling**

****

* Pooling reducing the spatial dimensions (Width x Height) of the Input Volume for the next Convolutional Layer. It does not affect the depth dimension of the Volume.
* The transformation is either performed by taking the maximum value from the values observable in the window (called ‘max pooling’), or by taking the average of the values. Max pooling has been favoured over others due to its better performance characteristics.
* also called downsampling

### **Step 4 - Normalization (ReLU in our case)**

**alt text**

Normalization (keep the math from breaking by turning all negative numbers to 0) (RELU) a stack of images becomes a stack of images with no negative values.

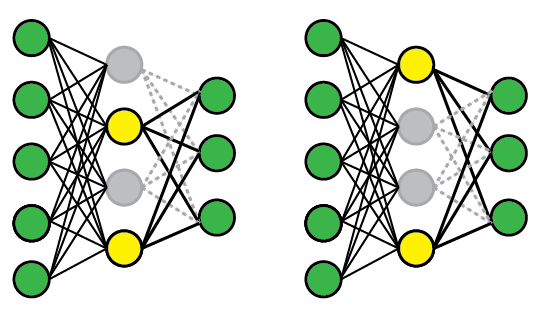
Repeat Steps 2-4 several times. More, smaller images (feature maps created at every layer)

### 

### **Step 5 - Regularization**

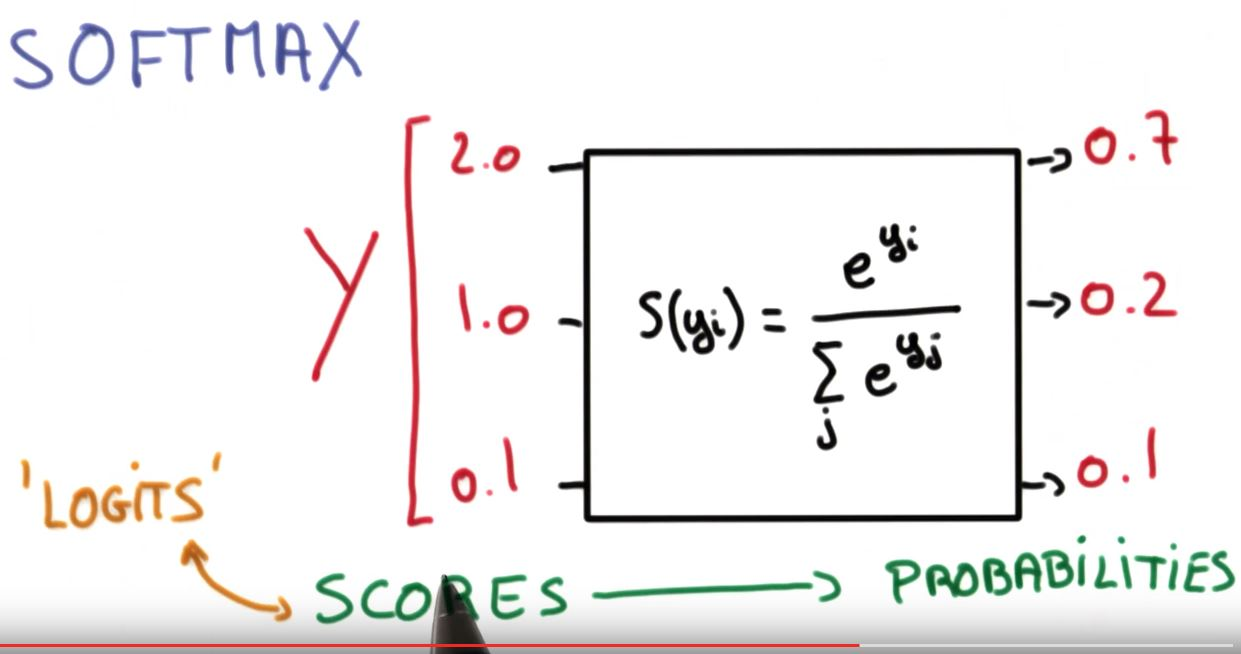
* Dropout forces an artificial neural network to learn multiple independent representations of the same data by alternately randomly disabling neurons in the learning phase.
* Dropout is a vital feature in almost every state-of-the-art neural network implementation.
* To perform dropout on a layer, you randomly set some of the layer's values to 0 during forward propagation.

See [this](http://iamtrask.github.io/2015/07/28/dropout/)



### **Step 6 - Probability Conversion**

At the very end of our network (the tail), we'll apply a Softmax function to convert the outputs to probability values for each class.



### **Step 7 - Choose most likely label (max probability value)**

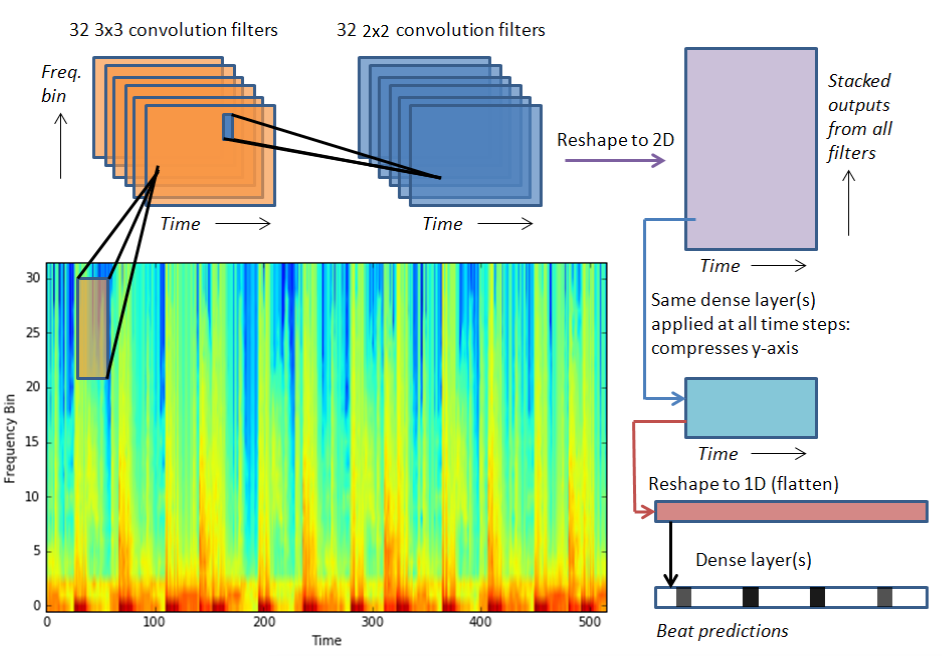
argmax(sigmoid\_outputs)

These 4 steps are one forward pass through the network.

## **When is a good time to use it?**

* To classify images
* To generate images (more on that later..)

Architecture of the Model implemented



**Model Architecture Design:**

classifier = Sequential()

classifier.add(Convolution2D(32, 3, 3, input\_shape = (150, 150, 3), activation ='relu'))

classifier.add(Dropout(0.1))

classifier.add(MaxPooling2D(pool\_size = (2,2)))

classifier.add(Convolution2D(64, 3, 3, input\_shape = (150, 150, 3), activation ='relu'))

classifier.add(Dropout(0.1))

classifier.add(MaxPooling2D(pool\_size = (2,2)))

classifier.add(Convolution2D(128, 3, 3, input\_shape = (150, 150, 3), activation ='relu'))

classifier.add(Dropout(0.1))

classifier.add(MaxPooling2D(pool\_size = (2,2)))

classifier.add(Flatten())

classifier.add(Dense(output\_dim = 128, activation = 'relu'))

classifier.add(Dropout(0.4))

classifier.add(Dense(output\_dim = 128, activation = 'relu'))

classifier.add(Dropout(0.4))

classifier.add(Dense(output\_dim = 6, activation = 'softmax'))

After the model is been built the and when the accuracy on train and validation is constant then model weights and graph architecture has to be saved.

# Fit the model and generate the classifier

classifier.fit\_generator(training\_set,

steps\_per\_epoch=16,

epochs=10,

validation\_data=validation\_generator,

validation\_steps=10)

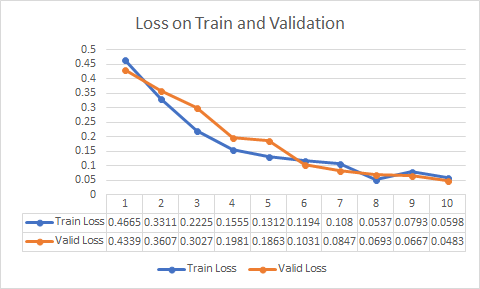
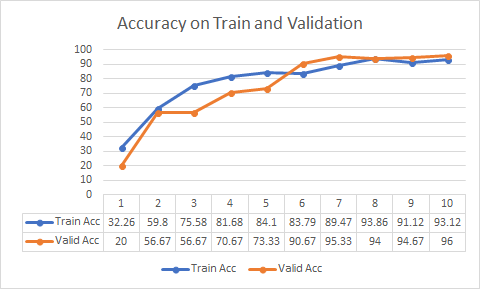
# serialize weights to HDF5 and save them for future references

classifier.save\_weights("model Conv2d.h5")

classifier.save('model.h5')

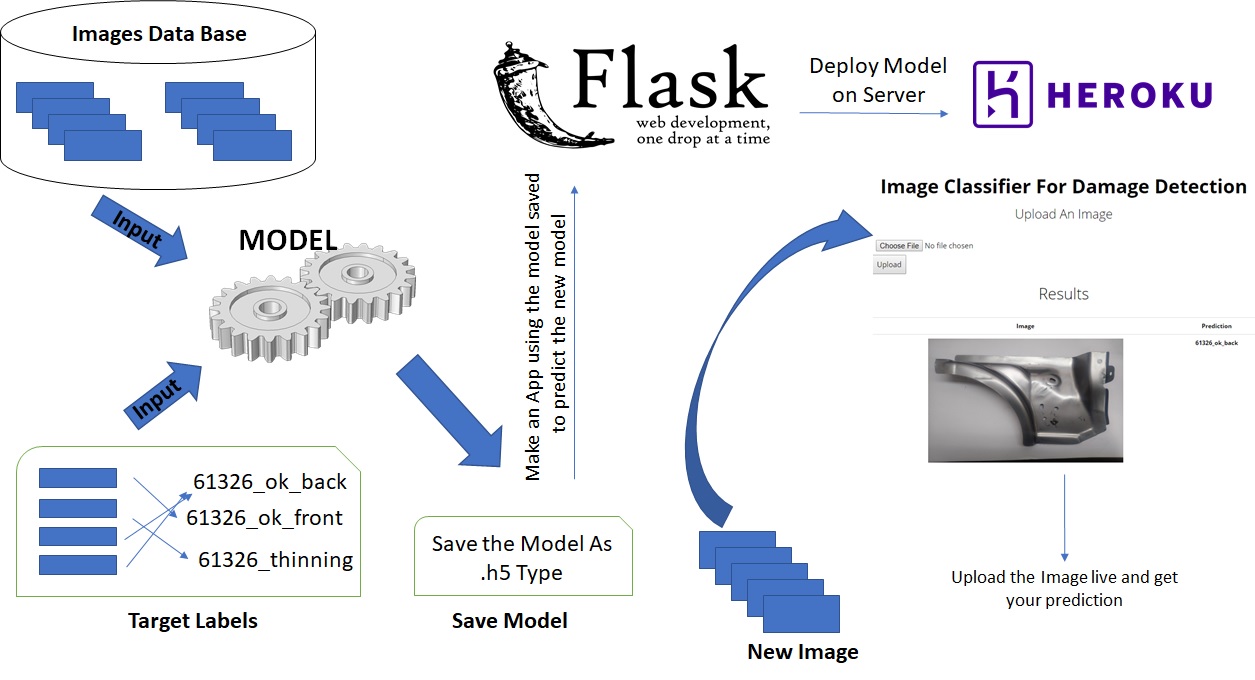
print("Saved model to disk")

The Train and Validation Accuracy and Loss are been given below as a chart:



From the above graph we can attain and highest accuracy of 96% on the validation data.

**Final Model Flow**



## 

## 

## 

## **Flask**

## 

## **What is Web Framework?**

Web Application Framework or simply Web Framework represents a collection of libraries and modules that enables a web application developer to write applications without having to bother about low-level details such as protocols, thread management etc.

## **What is Flask?**

Flask is a web application framework written in Python. It is developed by Armin Ronacher, who leads an international group of Python enthusiasts named Pocco. Flask is based on the Werkzeug WSGI toolkit and Jinja2 template engine. Both are Pocco projects.

Flow:

Use the model which was been saved as .h5 and define a function which takes the loaded model and use that as tool to predict the future images which comes into the environment.

Whenever there is requirement to train model again then run the **‘jbm\_model.py’** with the new data and store the new model and link that to app and make it live.

Run the app to check by routing the cmd to desired folder where the whole pack is there and run **‘app.py’.**

**Heroku**

Heroku is a new type of “webhost” and here is a tutorial on getting started. Heroku is different from your traditional shared web hosting such as BlueHost, HostGator, Dreamhost, etc. Heroku is a cloud platform as a service (PaaS) that supports several various programming languages such as nodejs, python, php, etc. You are billed based on the number of “dynos” you use. You no longer put files on the server using FTP, you use GIT.

Heroku deployment:

The following example demonstrates initializing a Git repository for an app that lives in the myapp directory:

$ cd myapp  
$ git init  
Initialized empty Git repository in .git/  
$ git add .  
$ git commit -m "My first commit"

## **[Creating a Heroku remote](https://devcenter.heroku.com/articles/git#creating-a-heroku-remote)**

Git [remotes](http://git-scm.com/book/en/Git-Basics-Working-with-Remotes) are versions of your repository that live on other servers. You deploy your app by pushing its code to a special Heroku-hosted remote that’s associated with your app.

### For a new Heroku app:

### The heroku create CLI command creates a new empty application on Heroku, along with an associated empty Git repository. If you run this command from your app’s root directory, the empty Heroku Git repository is automatically set as a remote for your local repository.

### $ heroku create

You can use the git remote command to confirm that a remote named heroku has been set for your app:

$ git remote -v

## **[Deploying code](https://devcenter.heroku.com/articles/git#deploying-code)**

To deploy your app to Heroku, you typically use the git push command to push the code from your local repository master branch to your heroku remote, like so:

$ git push heroku master

Use this same command whenever you want to deploy the latest committed version of your code to Heroku.

Note that Heroku only deploys code that you push to the master branch of the heroku remote. Pushing code to another branch of the remote has no effect.

**IV. Final Summary**

**Deploying The Model On Local Machine**

**Step1:**

Download the file in the repository:

Repository Link: <https://github.com/abhiyerasi/JBM-Assignment>

**Step 2:**

Open the Command Prompt:

Change the Directory were the file are there.

Then if need to train the model again:

Enter the command : **python jbm\_mode.py**

**Step 3:**

For running the app locally

Enter the Command: **python app.py**

**Step 4:**

After running the above commands once you get the link for local host:

Click on this link : <http://localhost:3000>

This steps are been used to deploy the model locally on machine.

**Deploying Live and To work on Mobile**

**Step 1:**

Create a new App and Give the new on Heroku Platform.

**Step 2:**

Check the deployment method as Git and connect the repository were the model and script are present which reflects the structure we need.

**Step 3:**

Click on the Manual Deploy version then automatically it will be hosted on the server.

Finally the new images has to be uploaded in the given url and the prediction will be given as output with the image.

The Url were the model is been hosted: <https://damage-detection-model.herokuapp.com>