

Image Inpainting Using Generative Adversarial Networks

Mounika Yamani
PG Scholar in CSE, CVRCE,
Hyderabad, India,
mounikayamani18@gmail.com

M. Raghava, Ph.D
Professor in CSE, CVRCE
Hyderabad, India
raghava.m@cvr.ac.in

ABSTRACT:

There is need to recover the damaged photographs, ancient paints, etc. Damage may occur due to several causes such as overlaid text, scratches, and scaled image to recover the image from such cases and providing a good looking photograph using a technique called inpainting. This term inpainting is also called as the observer does not know the original image. This inpainting has been done by professional artists. However, we did not get the pure accuracy and quality if it was done by manually and also more time-consuming process.

We are proposing a method to inpaint the images with higher efficiency by using a generator and discriminator networks known as GAN (Generative Adversarial networks). with the help GAN we can inpaint region by generating images with the help of generator, and discriminator to label the images. Here we evaluate the performance of inpainting by finding the G-Loss(generator loss) D-loss(discriminator loss), Mean squared error (MSE) and structural similarity of an image(SSIM).

INTRODUCTION:

Images get corrupted by random scratches during image acquisition process as the old photographs may get damaged due to cracks. Missing regions in the image is filled with the help of the inpainting

techniques. As early as the Renaissance, people try to restore damaged paintings in a way that the paintings will properly look like the original for an observer who is unfamiliar with the original. This process is called restoration, conservation, *Inpainting* or retouching. This is done to preserve the paintings and other fine art for future generations. example of image inpainting was shown in figure 1(a),1(b).



figure 1: Example of Image Restoration

1.1 (a) Original image with scratches

1.2(b) Restored image

Image inpainting is a widely used reconstruction technique by advanced photo and video editing applications for repairing damaged images or refilling the missing parts. The aim of the inpainting can be stated as reconstruction of an image without introducing noticeable changes. Although fixing small deteriorations are relatively simple, filling large holes or removing an object from the scene are still challenging due to huge variabilities and complexity in

the high dimensional image texture space. We propose a neural network model and a training framework that completes the large blanks in the images. As the damaged area(s) take up large space, hence the loss of information is considerable, the CNN model needs to deal with both local and global harmony and conformity to produce realistic outputs. Recent advances in generative models show that deep neural networks can synthesize realistic looking images remarkably, in applications such as super-resolution, deblurring, denoising and inpainting.

In this work we present an approach for inpainting using Generative adversarial network(GAN). The proposed method is capable of reconstructing images and at the same time provides higher resolution, i.e. we are able to inpaint blindly at higher resolution. convolutional neural network learns an end to end mapping between corrupted low resolution images and corresponding true high resolution images with little pre-processing. This reduces computational complexity and also do not require any prior information about missing region.

IMAGE INPAINTING Definition:

Image inpainting is a process to modify an image in a non-detectable form. The aim of inpainting algorithms is to recover missing information in an image(cracks in the old photographs, spots on an image) such that the resultant image is visually plausible.

PROBLEM DEFINITION: Given image with significant portions missing or damaged. Reconstitute missing regions with data consistent with the rest of the image. The core challenge of image inpainting lies in synthesizing visually realistic and semantically

plausible pixels for the missing regions that are coherent with existing ones.

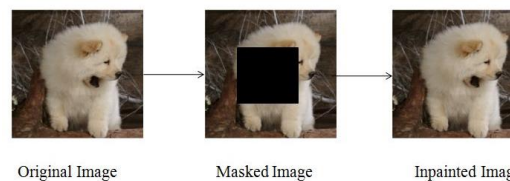


figure 2: Illustration of Problem Definition

LITERATURE SURVEY:

In the literature survey, we have included some of the previous techniques implemented for inpainting image restoration which we have referred to solve inpainting problem. Image Restoration techniques are majorly divided in to three categories. These comprises of following methods.

- (i) Partial Differential Equation (PDE) based
- (ii) Texture synthesis based
- (iii) Exemplar and search based

Partial Differential Equation (PDE) based : Partial Differential Equation (PDE) based algorithm is proposed by Marcelo Bertalmio et.al [1]. This algorithm is the iterative algorithm. The algorithm is to continue geometric and photometric information that arrives at the border of the occluded area into area itself. This is done by propagating the information in the direction of minimal change using isotope lines. This algorithm will produce good results if missed regions are small one. But when the missed regions are large this algorithm will take so long time and it will not produce good results.

Texture synthesis based : In Texture synthesis based inpainting, texture is synthesized in a pixel by pixel way, by picking existing pixels with similar neighbor

hoods in a randomized fashion. This algorithm performs very well but it is very slow since the filling-in is being done pixel by pixel. The texture synthesis [2] based Inpainting perform well in approximating textures. These algorithms have difficulty in handling natural images as they are composed of structures in form of edges. Hence while appreciating the use of texture synthesis techniques in Inpainting, it is important to understand that these methods address only a small subset of Inpainting issues and these methods are not suitable for a large objects.

Exemplar and search based: Third category is exemplar based inpainting. Criminisi et al. in [3] presented a technique for region filling and object removal by exemplar based inpainting. Using this algorithm both texture and structure are propagated in the missing area. The algorithm is based on patch based filling approach. The best matching patch(example) is found from known region and copied to the unknown region. Thus propagating the information. Although, the technique is superior than previously developed approaches in terms of both visual quality and computation efficiency, the limitation is that it fails to obtain reasonable results when similar patches are not found.

Dataset:

Dataset is set of data items used for applying our model to predict the result. In olden days we have used registers to store data, As technology is increasing and world is moving towards digitalization the usage of data is drastically improving day by day. We are no longer using registers, now a days organizations providing data in the form of excel sheets i.e in the form of CSV (Comma Separated values)or even in the

form of collection of Images with labels. We have used

CIFAR10 dataset: The **CIFAR-10 dataset** (Canadian Institute For Advanced Research) is a collection of images that are commonly used to train machine learning and computer vision algorithms. It is one of the most widely used datasets for machine learning research. The CIFAR-10 dataset contains 60,000 32x32 color images in 10 different classes. we can import CIFAR10 dataset in keras by using following line.

```
from keras.datasets import cifar10
```

PROPOSED GAN BASED MODEL:

GAN (Generative Adversarial Networks): GANs or Generative Adversarial Networks are Deep Neural Networks that are generative models of data. Given a set of training data, GANs can learn to estimate the underlying probability distribution of the data. This is very useful, because apart from other things, we can now generate samples from the learnt probability distribution that may not be present in the original training set.

Given a training set X (say a few thousand images of cats), The Generator Network, $G(x)$, takes as input a random vector and tries to produce images similar to those in the training set. A Discriminator network, $D(x)$, is a binary classifier that tries to distinguish between the real images according the training set X and the fake images generated by the Generator. As such, the job of the Generator network is to learn the distribution of the data in X , so that it can produce real looking images & make sure the Discriminator cannot distinguish between images from the training set and images from the Generator. The Discriminator needs to learn to keep up with the Generator trying new tricks all the time to

generate fake images and fool the Discriminator. Ultimately, the Generator learns the true distribution of the training data and becomes really good at generating real-looking images. The Discriminator can no longer distinguish between training set of images and generated images. In this way, two networks are continuously trying to make sure the other does not do a good job at their task.

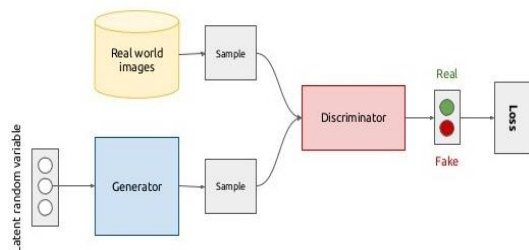


figure 3 : GAN concept: D tries to correctly detect, if images came from real world examples or G. G as the adversary tries to improve its image generation to fool D.

Generative Adversarial Network for Image Inpainting:

We introduce a generative model and a training procedure for the arbitrary and large hole filling problem. The generator network takes the corrupted image and tries to reconstruct the repaired image. We utilized the CNN architecture as our generator model with a few alterations. During the training, we employ the adversarial loss to obtain realistic looking outputs. The key point of our work is the following: we design model that uses discriminator to label the images that are generated by generator. The proposed System architecture is shown in Figure . Here we are using algorithm as GAN Model to train our dataset It compose of two networks:

1. Generator
2. Discriminator

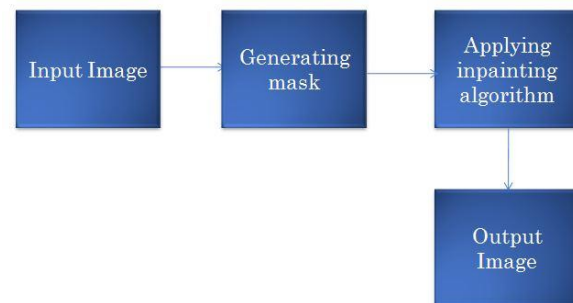


figure 4: System Architecture

Generator network: The generative network G takes the masked image from the user and forwards to the Discriminator to label the images. In our Project we have taken CIFAR10 dataset and categorize the images to cats and dogs and then labeling the images by passing through Discriminator which in turn undergoes Pool of Convolution layers ,after that model gets switched off for every 0.5% probability to not getting overfitting by using Drop-out layer. Here we have designed our generator to pass through 4 convolution layers and return the result to activation function which in turn returns to the model which generates the missing patch of the image and output gets returned to the Discriminator for labeling.

Discriminator network: Discriminator network D takes the generated real images and aims to distinguish them while the generator network G makes an effort to fool it. As long as D successfully classifies its input, G benefits from the gradient provided by the D network via its adversarial loss. We achieve our goal of obtaining an objective value that measures the quality of the image as a whole as well as the consistency in local details through our GAN approach depicted in Figure.

Rather than training two separate networks simultaneously, we design a weight sharing architecture at the first few layers so that they learn

common low level visual features. After a certain layer, they are split into two pathways. The first path ends up with a binary output which decides whether the whole image is real or not. The second path evaluates the local texture details similar to the GAN. Fully connected layers are added at the end of the second path of our discriminator network to reveal full dependency across the local patches.

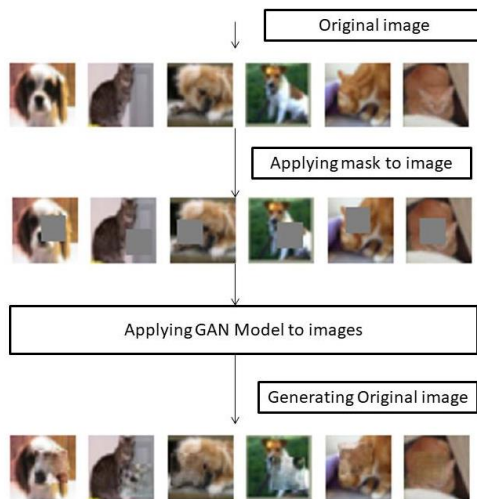


figure 5 : Applying GAN model to inpainting

The overall architecture hence provides an objective evaluation of the naturalness of the whole image as well as the coherence of the local texture. We evaluate our model by calculating two different losses known as L2 Loss with respect to generator as well as Discriminator point of view. In order to view the similarity between the images we use Structural Similarity Index of image as well as Mean Square Error.

Algorithm: Image Inpainting using GAN

Input: Supplying images from dataset

for number of training iterations do

 Sample half batch of images x

 Generating random mask m for x ;

 Construct inputs $x \cdot m$;

 Passing through pool of convolution layers

 returns bottle neck features

 Passing through deconvolution layer

 returns vector of $1 \times 1 \times n$

 Generator G generates images of $n \times n \times n$

 Discriminator D takes vector from Generator

 Categorize generated images \rightarrow valid and fake images;

 tries to reconstruct mask m that is missing of size $n \times n \times n$;

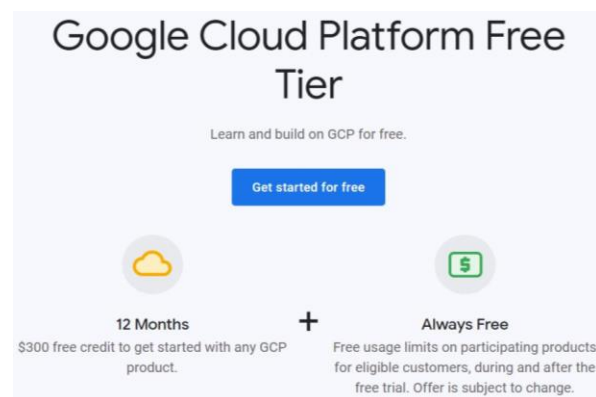
Terminates : By passing through Drop-out layer

Output: Mask generated

Experimental Setup

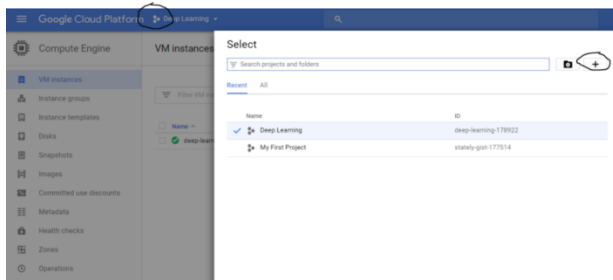
Jupyter Notebook Setup in Google Cloud Platform:

Step 1 : Creating Account: Create a free account in Google Cloud with 300\$ credit. For this step, you will have to put your payment (Credit card) information and verify your account.



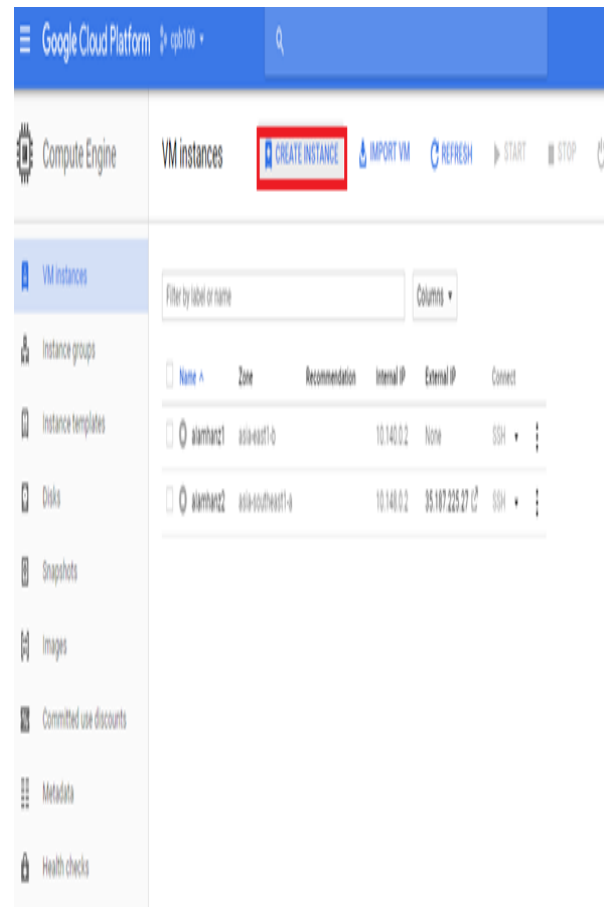
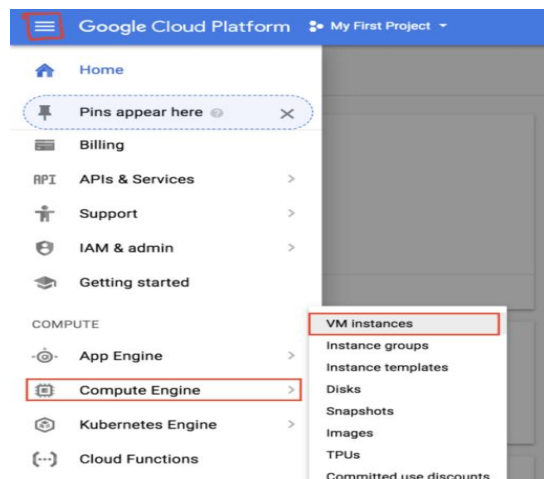
Step 2: Create a new project

- To use Jupyter Notebook on the instance, we need to create an instance first. Log into the Google Cloud Console. Create a new Project using the "+" sign as shown in the image.



Step 3 : Create VM Instance

- Once the project is created, select the project and click on the Products and icons and navigate to the **Compute Engine** label and click on **VM Instances**.
- Click **Create an instance** Name your instance, choose a zone that you want. Click on **Customize** and look for the appropriate memory, CPUs ,etc as per your need. Change Boot Disk accordingly and the SSD size.



Step 4: Configure the Virtual Machine

GCP provides various types of VM, with various CPU type, operating system and memory .Inorder to choose this Click on "**Customize**" to Customize as per your requirement As you can see on the picture, I'm using 8vCPUs with 30 GB Ram and the OS is Ubuntu 16.04 with 100 GB standard disk . As a note, the Zone choices is crucial because every different has different capability. Here we are Running this Project on GPU so Here we are choosing 1 GPU i.e. NVIDIA Tesla P100.

Google Cloud Platform My First Project

Compute Engine

VM Instances

Instance groups

Instance templates

Disks

Snapshots

Images

Create an instance

Name

Zone

Machine type 3.75 GB memory [Customize](#)

Name

Region

Zone

Machine type

Customize to select cores, memory and GPUs.

Cores vCPU 1-96

Memory GB 7.2-52

☐ Extend memory

CPU platform

GPUs

The number of GPU dies is linked to the number of CPU cores and memory selected for this instance. For the current configuration, you can select no fewer than 1 GPU die of this type. [Learn more](#)

Number of GPUs

GPU type

Boot disk

Select an image or snapshot to create a boot disk; or attach an existing disk

[OS images](#) [Application images](#) [Custom images](#) [Snapshots](#) [Existing disks](#)

- ☐ Debian GNU/Linux 8 (jessie)
amd64 built on 20180307
- ☐ Debian GNU/Linux 9 (stretch)
amd64 built on 20180307
- ☐ CentOS 6
x86_64 built on 20180314
- ☐ CentOS 7
x86_64 built on 20180314
- ☐ CoreOS alpha 1702.1.0
amd64-usr published on 2018-03-03
- ☐ CoreOS beta 1688.3.0
amd64-usr published on 2018-03-09
- ☐ CoreOS stable 1632.3.0
amd64-usr published on 2018-02-15
- ☐ Ubuntu 14.04 LTS
amd64 trusty image built on 2018-03-08
- ☒ Ubuntu 16.04 LTS
amd64 xenial image built on 2018-03-06
- ☐ Ubuntu 17.10
amd64 artful image built on 2018-03-14
- ☐ Container-Optimized OS 65-10323.55.0 beta
Kernel: ChromiumOS-4.4.111 Kubernetes: 1.8.7 Docker: 17.03.2
- ☐ Container-Optimized OS 66-10452.13.0 dev
Kernel: ChromiumOS-4.14.22 Kubernetes: 1.9.3 Docker: 17.03.2
- ☐ Container-Optimized OS 64-10176.62.0 stable
Kernel: ChromiumOS-4.4.96 Kubernetes: 1.8.7 Docker: 17.03.2
- ☐ Red Hat Enterprise Linux 6
x86_64 built on 20180314
- ☐ Red Hat Enterprise Linux 7
x86_64 built on 20180314
- ☐ Red Hat Enterprise Linux for SAP Applications 7.4
x86_64 built on 20180314
- ☐ Red Hat Enterprise Linux for SAP HANA 7.4

Can't find what you're looking for? Explore hundreds of VM solutions in [Cloud Launcher](#)

Boot disk type Size (GB)

- You will need to check off the HTTP traffic it will needed when we access the instance for Jupyter Notebook. You may un-check "Delete boot disk when instance is deleted" to ensure the data is available even after the instance dies.

Identity and API access

Service account

Access scopes

- ☒ Allow default access
- ☐ Allow full access to all Cloud APIs
- ☐ Set access for each API

Firewall

Add tags and firewall rules to allow specific network traffic from the Internet

- ☒ Allow HTTP traffic
- ☒ Allow HTTPS traffic

Management [Disks](#) [Networking](#) [SSH Keys](#)

Deletion rule

- ☐ Delete boot disk when instance is deleted

And click on Create. You are done!! The instance will be up and running in a few minutes.

Name	Zone	Recommendation	In use by	Internal IP	External IP	Connect
Instance-1	us-east1-b			10.142.0.2 (nic0)	None	SSH
Instance-2	us-east1-b			10.142.0.3 (nic0)	None	SSH
Instance-3	us-east1-b			10.142.0.4 (nic0)	104.196.48.190	SSH

Step 7: Start your VM instance

Now start your VM instance. When you see the green tick click on SSH. This will open a command window and now you are inside the VM.

Name	Zone	Recommendation	In use by	Internal IP	External IP	Connect
Instance-1	us-east1-b			10.142.0.2 (nic0)	None	SSH
Instance-2	us-east1-b			10.142.0.3 (nic0)	35.243.237.238	SSH
project	us-west1-b			10.138.0.2 (nic0)	34.83.192.112	SSH

- Let us move in to the shell of the instance we just created. To do that, it is very simple in google cloud. They have a shell which can be accessed over the browser. This saves us from the hassle of accessing the SSH(Secure Shell) keys.
- Let us go back to the instance page and Select the instance and against the SSH value, there is a drop down list with 4 options. We will use the first one. As soon you login to the shell, run password and make a note of your user directory. That will be needed for the rest of the steps.

project	us-west1-b	10.138.0.2	34.83.192.112	SSH
---------	------------	------------	---------------	-----

- Then, the terminal pops-up like as shown in below figure.

```
srikanthyamani5@instance-3: ~ - Google Chrome
https://ssh.cloud.google.com/projects/molten-infusion-242205/zones/us-east1-b/in
Connected, host fingerprint: ssh-rsa 0 E6:24:6F:73:63:A7:4D:ED:6F:EF:7
:2E:D4:9C:BF:04:D5:D2:06:22:DA:15:DA:63:C3:58:BF:45:10
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.15.0-1033-gcp x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

0 packages can be updated.
0 updates are security updates.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

srikanthyamani5@instance-3:~$
```

Step 8 : Install Jupyter notebook and other packages

In your SSH terminal, enter:

```
wget http://repo.continuum.io/archive/Anaconda3-4.0.0-
Linux-x86_64.sh
```

This will install anaconda and all the dependencies on the Linux environment.

```
source ~/.bashrc
```

Now, install other softwares :

```
pip install keras
```

Step 10 : Launching Jupyter Notebook

To run the Jupyter notebook, just type the following command in the ssh window you are in :

```
Jupyter-notebook --no-browser --port=<PORT-NUMBER>
```

Once you run the command, it should show something like this:

Here the port number is 5000 which we have given in step 5.

```
~$ vim /home/srikanthyamani5/.jupyter/jupyter_notebook
~$ jupyter-notebook --no-browser --port=5000
Writing notebook server cookie secret to /run/user/10
```


In order to launch your Jupyter notebook, just type the following in your browser:

`http://<External Static IP Address>:<Port Number>`

where, external ip address is the ip address which we made static and port number is the one which we allowed firewall access to.

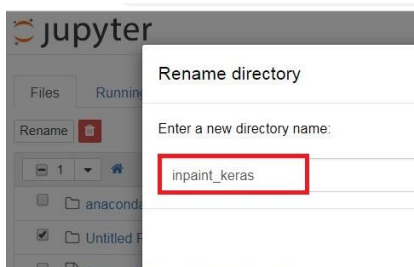
Here ip address is: 104.196.48.190



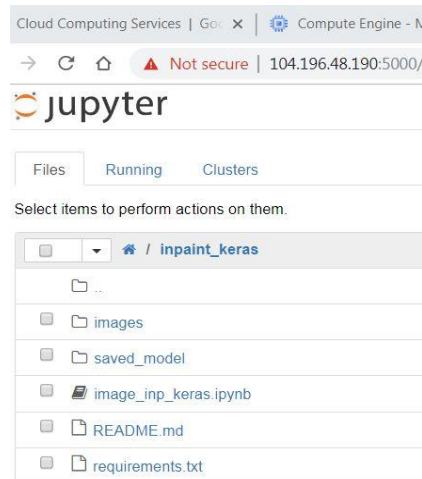
Step 12 : Creating Project folder in Jupyter Notebook

In order to create folder in Jupyter notebook on the right corner of the notebook there is “new” button on clicking on “new “ list is pop up on the screen with four options , among them select Folder to create a folder.

Here we are naming Project folder as **inpaint_keras** as shown in below.



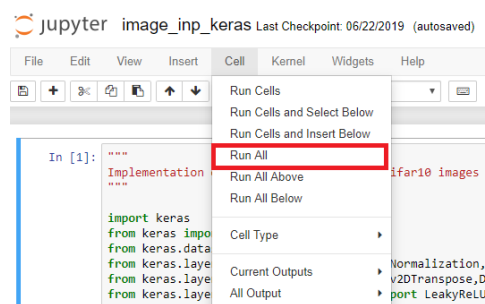
After uploading the project files to the Project folder in **inpaint_keras**, it looks like as shown in below figure:



In the above figure **image_inp_keras.ipynb** is a python file in which project code is written, **saved_model** is sub folder in project folder where images are stored during the execution of code at respective epochs. (iterations).

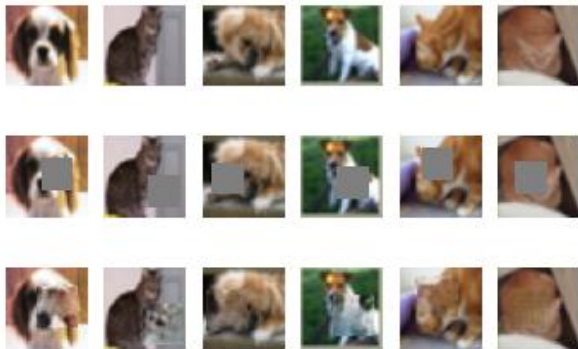
Step 13 : Running Project in Jupyter Notebook:

In order to run the Project in Jupyter Notebook go to Cell->Run All to run the Project.



After running the Project each cell gets executed at an equal number of epochs. Following is the figure which shows each and every epoch and their loss and accuracy levels of inpainting the image.

9981	[D loss: 0.102076, acc: 100.00%]	[G loss: 0.084149, mse: 0.078867, acc: 73.95%]
9982	[D loss: 0.001554, acc: 100.00%]	[G loss: 0.079797, mse: 0.075564, acc: 66.28%]
9983	[D loss: 0.047137, acc: 100.00%]	[G loss: 0.065421, mse: 0.058595, acc: 68.59%]
9984	[D loss: 0.074135, acc: 96.88%]	[G loss: 0.100895, mse: 0.096506, acc: 70.43%]
9985	[D loss: 0.951044, acc: 62.50%]	[G loss: 0.078025, mse: 0.075363, acc: 70.46%]
9986	[D loss: 0.043238, acc: 100.00%]	[G loss: 0.087379, mse: 0.086392, acc: 63.42%]
9987	[D loss: 0.018257, acc: 100.00%]	[G loss: 0.075969, mse: 0.074694, acc: 68.54%]
9988	[D loss: 0.083331, acc: 93.75%]	[G loss: 0.065471, mse: 0.063483, acc: 68.15%]
9989	[D loss: 0.004256, acc: 100.00%]	[G loss: 0.083141, mse: 0.079747, acc: 67.81%]
9990	[D loss: 0.008722, acc: 100.00%]	[G loss: 0.073262, mse: 0.069885, acc: 73.48%]
9991	[D loss: 0.016486, acc: 100.00%]	[G loss: 0.078637, mse: 0.067635, acc: 73.43%]
9992	[D loss: 0.002798, acc: 100.00%]	[G loss: 0.064434, mse: 0.062734, acc: 70.21%]
9993	[D loss: 0.444954, acc: 78.12%]	[G loss: 0.071113, mse: 0.063873, acc: 72.25%]
9994	[D loss: 0.007060, acc: 100.00%]	[G loss: 0.072429, mse: 0.065976, acc: 75.06%]
9995	[D loss: 0.091749, acc: 96.88%]	[G loss: 0.082295, mse: 0.075510, acc: 69.21%]
9996	[D loss: 0.029383, acc: 100.00%]	[G loss: 0.078998, mse: 0.064536, acc: 71.55%]
9997	[D loss: 0.074465, acc: 96.88%]	[G loss: 0.084626, mse: 0.077886, acc: 69.66%]
9998	[D loss: 0.012157, acc: 100.00%]	[G loss: 0.077282, mse: 0.073597, acc: 68.33%]
9999	[D loss: 0.024736, acc: 100.00%]	[G loss: 0.069424, mse: 0.066433, acc: 69.02%]



CONCLUSION:

Image inpainting using Generative Adversarial Networks uses generator and discriminator as two networks to fill the missing parts of the images, instead of using only the CNN independently which increase computation time more and it passes through several CNN layers to get the resemble output, we have defined a model with four convolution layers and a pool and dropout layer, after series of convolutional layers we applied deconvolution layers(autoencoder), then we apply activation function to get the model as an output, which passes through discriminator which is used to build the patch(64*64) that is missing in the image with the help of Generator. In this way we have inpainted the region of an images(128*128 in size) of dataset i.e CIFAR10

References:

- [1] Marcelo Bertalmio, Luminata Vese, Guillermo Sapiro (2003), "Simultaneous Structure and Texture Image In painting", *IEEE transactions on image processing*, vol. 12
- [2] K. Sangeetha, Dr. P. Sengottuvelan, E. Balamurugan,"Combined Structure and Texture Image Inpainting Algorithm for Natural Scene Image Completion." - Journal of Information Engineering and Applications. ISSN 2224-5758 (print) ISSN 2224-896X Vol 1, No.1, 2011.
- [3] Antonio Criminisi, Patrick Pérez, and Kentaro Toyama. Region filling and object removal by exemplar-based image inpainting. *Image Processing, IEEE Transactions on*, 13(9):1200–1212, 2004.
- [4] N.Neelimal, M.Arulvan,"Object Removal by Region Based Filling Inpainting",978-1-4673-5301-4/13,IEEE,2013.
- [5] Kohler, R., Schuler, C., Scholkopf, B., Harmeling, S.: Mask-specific inpainting with deep neural networks. In: *German Conference on Pattern Recognition*, Springer(2014) 523-534
- [6] Xie, J., Xu, L., Chen, E.: Image denoising and inpainting with deep neural networks. In: *Advances in Neural Information Processing Systems*. (2012) 341-349
- [7] N.Neelimal, M.Arulvan,"Object Removal by Region Based Filling Inpainting",978-1-4673-5301-4/13,IEEE,2013.



Mounika Yamani is pursuing his Masters of Technology in CVR College of Engineering Hyderabad, India in Computer Science and Engineering specialization. she will complete

her PG in 2019.

M.Raghava is working as Associate Professor in
Department of Computer Science and Engineering at
CVR College of Engineering, Hyderabad.