IMAGE INPAINTING

Project Report

ProjectX Mentorship Programme

At
Community of Coders,
Veermata Jijabai Technological Institute,Mumbai
November 2023

Acknowledgement

I am truly honored to express my sincere appreciation and gratitude to the individuals and organizations that have made my journey through the Project-X Mentorship Programme, a remarkable and enlightening experience.

First and foremost, I extend my heartfelt thanks to my mentors, Dhruvanshu Joshi and Om Doiphode, whose unwavering support, guidance, and expertise have been instrumental in shaping the success of this project. Their mentorship not only provided invaluable knowledge but also instilled in me a sense of confidence and passion for the subject matter.

I would also like to acknowledge the Community of Coders,VJTI Mumbai for their dedication to fostering the growth and development of aspiring professionals through this mentorship initiative. Their commitment to education and professional development is commendable.

My fellow participants in the program have been a source of inspiration, motivation, and camaraderie. Their collaboration and shared experiences have enriched my learning journey.

Lastly, I want to thank all the other mentors, educators, and supporters who, in various ways, contributed to the success of this project and my overall growth.

This mentorship program has not only expanded my knowledge and skills but also broadened my perspective on my chosen field. I am excited to carry forward the valuable lessons learned during this journey and apply them to my future endeavors.

Aditi Dhumal
aditi.dhumal1803@gmail.com
+91 9820966328
Kindipsingh Mallhi
kindip.mallhi@gmail.com
+91 8779907507

TABLE OF CONTENTS:

1. PROJECT OVERVIEW

1.1 Description:

In the realm of computer vision and image processing, our image inpainting project takes center stage as a creative and innovative solution for the restoration and enhancement of digital images. Image inpainting is the art of intelligently filling in missing or damaged regions within an image, and our project leverages state-of-the-art deep learning techniques to achieve this. With a focus on preserving the visual consistency and naturalness of the image, our model utilizes convolutional neural networks and advanced inpainting methods to reconstruct missing content seamlessly. Whether it's restoring old photographs, removing unwanted objects from images, or enhancing visual storytelling by completing missing parts of a scene, our image inpainting project represents a versatile and powerful tool for various applications. It brings forth a new dimension in image editing and restoration, providing users with the means to effortlessly repair and enhance their visual content while maintaining the authenticity and aesthetics of the original images.

This description highlights the significance and versatility of image inpainting, emphasizing the use of advanced deep learning techniques to achieve visually pleasing results while preserving the original image's essence.

1.2 Technologies That Made This Project Possible:

- **OpenCV (Open Source Computer Vision Library) https://opencv.org/:** OpenCV served as the foundational framework for computer vision tasks in our project. It provided a rich set of tools and functions for image processing, feature extraction, and various computer vision operations. Its versatility and wide adoption made it an indispensable resource.
- **NumPy (Numerical Python) https://numpy.org/:** NumPy was the backbone of our project for numerical computations and array manipulations. This library, designed for scientific computing in Python, facilitated the handling of complex data structures and performed mathematical operations efficiently. Its array-based approach simplified data manipulation.
- **Matplotlib (Data Visualization Toolkit) https://matplotlib.org/:** Matplotlib empowered our project with extensive data visualization capabilities. This versatile library enabled the creation of static and interactive visualizations, making it a crucial tool for presenting and analyzing data effectively. It offered a wide range of customization options.
- **Google Collaboratory (Colab) https://colab.research.google.com/:** Google Colaboratory served as our cloud-based development environment, providing access to powerful GPUs and TPUs. This collaborative platform facilitated real-time collaboration on code and accelerated machine learning tasks. Its integration with Google Drive simplified data sharing and storage.
- **TensorFlow https://www.tensorflow.org/:** TensorFlow, an open-source machine learning framework by Google, played a central role in building and training machine learning models. Its comprehensive ecosystem supported deep learning and neural network development, offering tools and libraries for various machine learning tasks.
- **PyTorch https://pytorch.org/:** PyTorch, a dynamic deep learning framework developed by Facebook, offered flexibility and ease of use for our deep learning experiments. Its dynamic computation graph allowed for rapid prototyping of neural networks, making it a valuable choice for research and experimentation.
- **Kaggle https://www.kaggle.com/:** Kaggle, a prominent data science platform, served as a valuable resource for datasets, competitions, and a thriving data science community. It provided access to diverse datasets,

kernels for code sharing, and a platform for hosting machine learning competitions. The Kaggle community fostered knowledge sharing and collaboration.

- **Weights and Biases (Wandb) https://wandb.ai/:** Weights and Biases, often referred to as Wandb, played a critical role in experiment tracking and visualization. This platform allowed us to monitor training progress, evaluate model performance, and visualize project metrics effectively. It enhanced the reproducibility and transparency of our experiments.

2. Introduction:

2.1 General:

The core objective of our project is to showcase the application of image inpainting through two distinct techniques.
Firstly, we explore the traditional and well-established method known as the Fast Marching Method (FMM). This method predates the era of deep learning and relies on a weighted average approach to determine the intensity of pixels to be inpainted. The heart of FMM lies in solving the eikonal equation, a fundamental step in this process. Our implementation of FMM draws inspiration from a seminal paper mentioned in the resources. :that serves as our reference point for this technique.

In contrast, we delve into more advanced techniques by incorporating deep learning, specifically through the use of Partial Convolutions.
This modern approach involves the construction of a neural network model using the Keras framework. Our experimentation with Partial Convolutions was conducted on the Tiny ImageNet 200 dataset.
To train our model, we leveraged the resources available on Kaggle. Throughout the training process and for monitoring and tracking various aspects of our experiments, we employed Weights and Biases (Wandb), enhancing the transparency and reproducibility of our work.

2.2 Basic Project Domains:

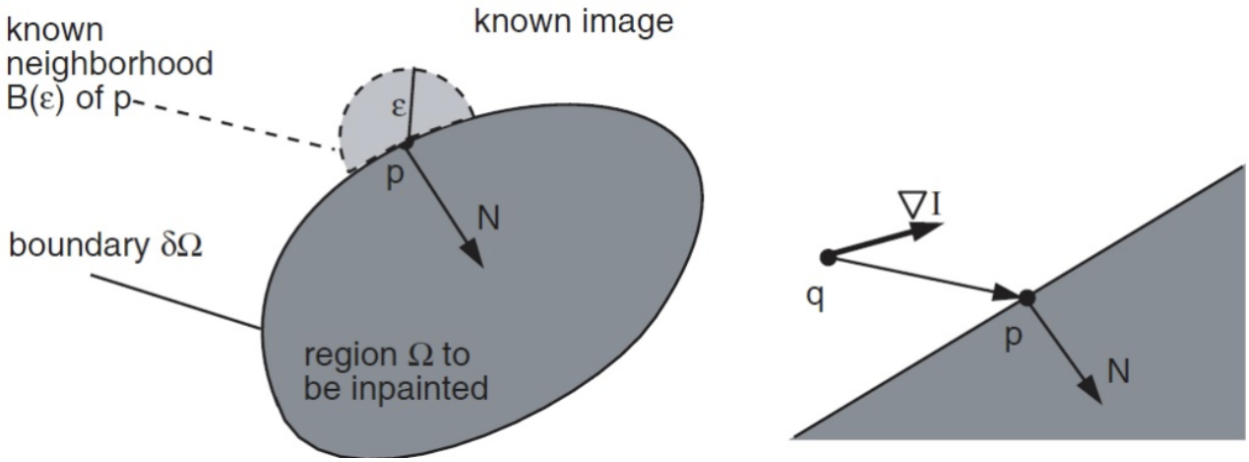
- OpenCV
- Matplotlib
- Tensorflow
- Numpy
- Keras

2.3 Theory:

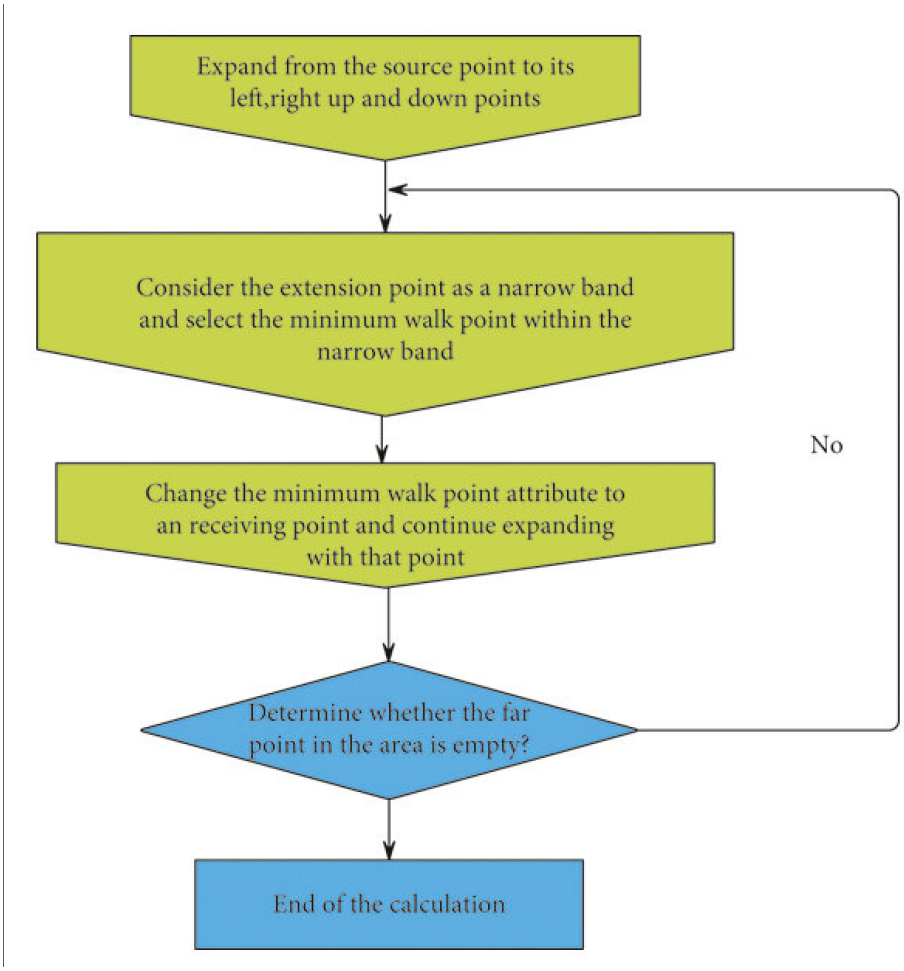
2.3.1 Fast Marching Method:

The inpainting algorithm systematically restores images by filling in missing or damaged regions. It starts by creating a binary mask, which acts as a guide for inpainting. The mask designates areas to be inpainted (typically marked as black), while untouched areas are kept white.
The algorithm then employs a modified Fast Marching Method (FMM) to process the image. It classifies pixels into three categories: Known (areas with pixel values available), Band (near the known regions), and Inside (unknown areas). The primary goal is to estimate pixel values in the Inside region based on neighboring Known pixels.
Here's the step-by-step process:
Binary Mask:

- Create a binary mask M, where M(i, j) is 1 for known regions and 0 for areas to be inpainted.



Fast Marching Method (FMM):



- Categorize pixels into three regions: Known (K), Band (B), and Inside (I).
- Initialize the distance function T(i, j) as follows:
- T(i, j) = 0 for K
- T(i, j) = ∞ for B
- T(i, j) = ∞ for I

Iterative Pixel Estimation (for each pixel (i, j) in I):

- Calculate T(i, j) by minimizing the Eikonal equation:
- $\nabla T \cdot \nabla T = 1$
- Update T(i, j) based on the minimum value of neighboring pixels T(i+1, j), T(i-1, j), T(i, j+1), T(i, j-1).

Pixel Value Estimation:

- Estimate pixel values using gradient information, spatial distance, and similarity:
- I(i, j) = (Σw * I_neigh) / Σw
- w = (dot(∇T, r) / |r|) * (1 / (1 + |T_neigh - T(i, j)|))

Filtering:

- Apply filtering techniques (e.g., median, bilateral) to improve inpainting quality and reduce noise.

2.3.2 Partial Convolutions:

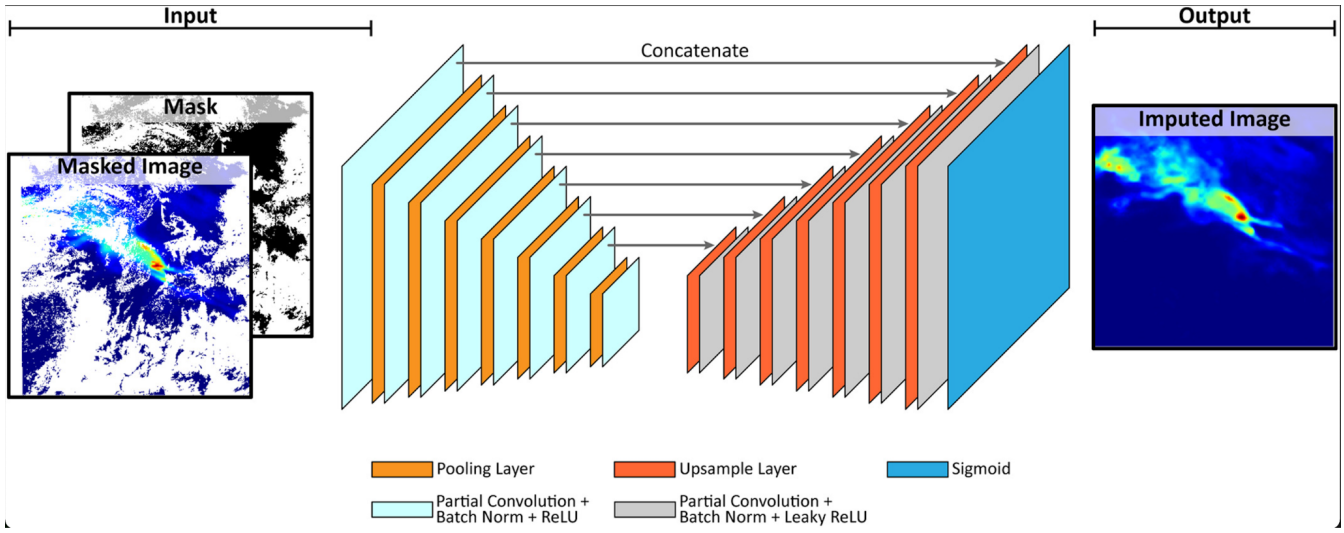


Image inpainting using Partial Convolutions is an advanced technique designed to restore missing or corrupted parts of an image. This method is particularly adept at handling complex datasets like Tiny ImageNet, due to its capability to learn rich representations from a vast variety of visual features and contexts.

In the case of the architecture provided, Partial Convolutions operate by applying a binary mask along with the input image to the convolutional network. The binary mask differentiates between the known (valid) and the unknown (invalid) pixels. Here's how the process unfolds:

Binary Mask Creation: A mask M is generated, where

M(i,j)=1 for known pixels and

M(i,j)=0 for pixels that need to be inpainted.

Partial Convolution: During this step, for every convolution operation, the input image I is element-wise multiplied by the maskM. The masked image is then convolved with a filter weight W. The result of this convolution is

normalized by the sum of the mask within the convolution window to maintain balance between the known and unknown regions:
$PConv(I,M)=W \cdot (I \odot M)/\sum(M)$
This equation ensures that the convolution is only applied to the valid pixels, and the normalization accounts for the different amounts of valid information within the kernel's receptive field.

Mask Update: Post convolution, the mask is also updated to reflect the changes in known and unknown pixel regions, enabling dynamic adaptation as the network iterates.

Iterative Refinement: The network uses the results of the partial convolutions to iteratively refine the inpainting results. It learns to predict the missing information by leveraging the context from the surrounding known pixels.

Learning and Optimization: The model undergoes training where it minimizes a loss function that encourages consistency between the reconstructed image and the original image in non-masked regions. Through backpropagation, the network's parameters are optimized, allowing it to learn how to fill in the missing parts coherently.

In the realm of digital image restoration, the methodical progression from traditional inpainting techniques to advanced deep learning approaches marks a transformative era. Our methodology encompasses setting up computational environments, leveraging algorithms like Fast Marching Method and partial convolutions, and iteratively refining models to adeptly reconstruct missing or damaged image segments.

3. Methods and Stages of Progress:

3.1 Prerequisites:

- Install numpy
- Login to Collab
- Sign in to wandb.ai
- Install matplotlib
- Install Open cv-python
- Install Keras

3.2 FMM Implementation:

3.2.1 Overview:

- FMM is a traditional image inpainting technique used to restore missing or damaged image regions.
- It works by propagating information from known pixel values to inpaint the unknown regions.
- The core idea is to create a "narrow band" of pixels that lie between the known and inpaintable areas.

The Narrow Band:

- In FMM, the narrow band is a region that separates the known (reliable) pixel values from the completely unknown pixels.
- It's a buffer zone that gradually shrinks as the inpainting process progresses.
- Pixels within the narrow band are candidates for inpainting, and their values are estimated based on neighboring known pixels.

Core Data Structures:

- FMM utilizes two key data structures:
- T (Distance Field Vector): This represents the distance from each pixel to the nearest known pixel.
- f (Flags): Flags are used to classify each pixel as Known, Band, or Inside (Unknown).

Propagation Algorithm:

- FMM employs a propagation algorithm to iteratively inpaint the unknown pixels based on neighboring known pixels.
- Pixels within the narrow band are considered for inpainting.

Code Snippet:

```
def solve(i1, j1, i2, j2, T, f):
  # ... (Initialization and boundary checks)
    # Check if both neighboring pixels are known
    if flag1 == KNOWN and flag2 == KNOWN:
        # Calculate pixel intensity based on known neighbors
        T1 = T[i1, j1]
        T2 = T[i2, j2]
        d = 2.0 - (T1 - T2) ** 2
        if d > 0.0:
            r = math.sqrt(d)
            s = (T1 + T2 - r) / 2.0
            if s >= T1 and s >= T2:
                return s
            else:
                s += r
                if s <= T1 and s <= T2:
                    return s
    # Handle cases where one or both neighbors are known
    if flag1 == KNOWN:
        T1 = T[i1, j1]
        return 1.0 + T1
    elif flag2 == KNOWN:
        T2 = T[i2, j2]
        return 1.0 + T2


    return sol
```

Inpainting Process:

- The inpaint function takes a pixel (i, j) within the narrow band and estimates its value using neighboring known pixels.

Code Snippet:

```
def inpaint(i, j, T, image):
    # ... (Initialization and gradient calculations)

    for z in range(3):  # Loop over color channels
        Ia = 0
        s = 0
        for x in Be:
            if 0 <= x[0] < image.shape[0] and 0 <= x[1] < image.shape[1] and f[x[0], x[1]] == KNOWN:
                # Calculate weights based on gradient and intensity differences
                # ...
                # Weighted averaging to estimate pixel value
                Ia += w * (image[x[0], x[1], z] + dot_gradi_r)
                s += w
        if s != 0:
            image[i, j, z] = Ia / s
        else:
            image[i, j, z] = 0
    return image
```

Propagation and Narrow Band Update:

- The for loop in the code snippet iteratively updates the narrow band by considering neighboring pixels.
- The T values are updated based on neighboring known pixels.
- Pixels are classified into Known, Band, or Inside categories based on f.

Code Snippet:

```
while narrow_band:
        min_distance = 1000000
        min_point = None

        for point in narrow_band:
            if T[point[0], point[1]] < min_distance:
                min_distance = T[point[0], point[1]]
                min_point = point

        current_point = min_point

        if current_point is None:
            break
        f[current_point[0],current_point[1]]=KNOWN
        narrow_band.remove(current_point)
```

```
       neighbors = [(current_point[0] - 1,
current_point[1]),(current_point[0] + 1,
current_point[1]),(current_point[0], current_point[1] -
1),(current_point[0], current_point[1] + 1)]

       for neighbor in neighbors:
         if 0 <= neighbor[0] < image.shape[0] and 0 <= neighbor[1] <
image.shape[1]:
           if f[neighbor[0],neighbor[1]]!=KNOWN:
             if f[neighbor[0],neighbor[1]]==INSIDE:
               f[neighbor[0],neighbor[1]]=BAND
               b=10
               inpaint(neighbor[0],neighbor[1],T,image)



T[neighbor[0],neighbor[1]]=min(solve(neighbor[0]-1,neighbor[1],neighbor
[0],neighbor[1]-1,T,f),solve(neighbor[0]+1,neighbor[1],neighbor[0],neig
hbor[1]-1,T,f),solve(neighbor[0]-1,neighbor[1],neighbor[0],neighbor[1]+
1,T,f),solve(neighbor[0]+1,neighbor[1],neighbor[0],neighbor[1]+1,T,f))
             narrow_band.append(neighbor)
```

3.2.2 Weights in FMM for Image Inpainting:

- Weights play a crucial role in determining the contribution of neighboring known pixels to the inpainting of an unknown pixel.
- These weights are calculated based on various factors, including gradient information, intensity differences, and the distance between pixels.

Gradient-Based Weighting:

- In the inpainting process, the gradient of the distance field vector (grad_T) is computed.
- The dot product of the gradient and the relative position vector (r) is used to calculate the direction (dir) of maximum change in distance.

Code Snippet:

```
dot_gradt_r = (grad_T[0][i, j] * r[0]) + (grad_T[1][i, j] * r[1])
dir = dot_gradt_r / (np.linalg.norm(r, 1))
if dir == 0:
    dir = EPS
```

- The dir represents the direction along which known pixels are expected to influence the unknown pixel.

Distance-Based Weighting:

- The distance between the unknown pixel and its neighboring known pixel is also considered.
- A weight factor (lev) is calculated inversely proportional to the absolute difference in distance (T[x[0], x[1]] - T[i, j]).

Code Snippet:

```
lev = 1 / (1 + np.abs(T[x[0], x[1]] - T[i, j]))
```

- lev reflects the influence of a known pixel on the unknown pixel based on their relative distances.

Overall Weight Calculation:

- The weights are calculated by combining the gradient-based direction (dir), distance-based influence (lev), and a term that considers the distance between the unknown pixel and its neighbors.
- These weights are then used to perform a weighted averaging of neighboring known pixel intensities, which is used to estimate the unknown pixel's value.

Code Snippet (Weighted Averaging):

```
w = (dir * dst * lev)
Ia += w * (image[x[0], x[1], z] + dot_gradi_r)
s += w
```

Final Image Enhancement:

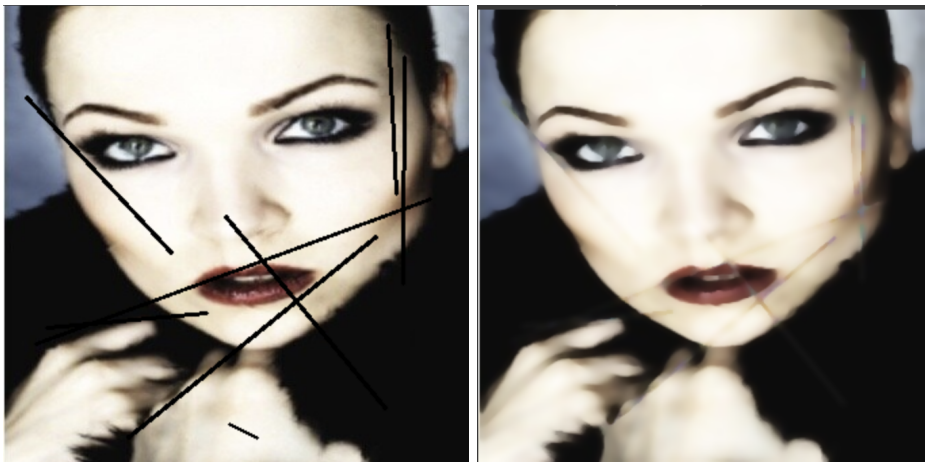
- The code further applies image enhancements such as median blur and bilateral filtering to improve the inpainted image's visual quality.

3.2.3 The Results using FMM:

The initial set of images comprises the input image, where the damaged pixels are delineated in black, and the inpainted result. It is worth noting that the damaged regions have been seamlessly restored with a thickness of 1 pixel. Notably, the inpainting process has yielded exceptionally satisfactory outcomes, exhibiting no discernible issues in the restored areas.

a) Input Image b) Inpainted Image

The second set of images illustrates that as the thickness of the damaged parts increases, the accuracy of the inpainting results diminishes. In these cases, the inpainted regions become more noticeable and stand out prominently. This highlights the need for improved methods that incorporate deep learning techniques to achieve more satisfactory inpainting outcomes.



a) Input Image b) Inpainted image

3.3 Partial Convolutions Implementation:

3.3.1 Overview of Partial Convolution

Partial convolution is designed for handling images with missing data by re-weighting the convolution based on the ratio of valid pixels at each convolution window. This approach allows a model to essentially "ignore" the missing pixels and make predictions solely based on the available information.

3.3.2 Steps Involved:

Step 1: Applying the Mask
In the provided code, the partial convolution starts with the application of the mask to the image:

```
output_img = K.conv2d(images*masks, self.kernel, ...)
```

Here's what happens:

images*masks: Each image tensor is multiplied element-wise by a corresponding binary mask tensor. In this binary mask, "1" represents valid pixels and "0" represents missing pixels or the area to be inpainted. This operation effectively zeroes out the areas of the image that are missing.
K.conv2d(...): The element-wise multiplied result is then convolved with a kernel self.kernel. Only the unmasked (valid) pixels are convolved since the masked pixels are set to zero.

Step 2: Convolution with the Kernel
During convolution, the kernel moves across the input image and performs an element-wise multiplication with the part of the image it covers, and then sums up the result. In standard convolution, all inputs are considered equal, but in partial convolution, the masked inputs are ignored.

Step 3: Updating the Mask
Simultaneously with the image convolution, the mask itself is convolved:

```
output_mask = K.conv2d(masks, self.kernel_mask, ...)
```

This operation updates the mask for the next layer, determining which pixels will be considered in the next partial convolution layer.

Step 4: Normalizing the Convolution Output
After convolution, the code normalizes the output based on how many valid pixels were in each window:

```
mask_ratio = self.window_size / (output_mask + 1e-8)
mask_ratio = K.clip(mask_ratio, 0, 1)
output_img *= mask_ratio
```

Here's what these lines do:

self.window_size / (output_mask + 1e-8): This computes the ratio of the number of valid pixels to the total number of pixels in the convolution window. The 1e-8 prevents division by zero.
K.clip(mask_ratio, 0, 1): Ensures that the mask ratio is within the range [0, 1].
output_img *= mask_ratio: The output of the convolution is then multiplied by this mask ratio. This step normalizes the contribution of each convolution window, ensuring that the contribution of windows with fewer valid pixels isn't unfairly downscaled.

Step 5: Repeated Application and Learning
In a neural network, these steps are applied repeatedly across multiple layers. During the training phase, the network learns the optimal weights for the convolution kernels by backpropagating the error between the network's output and the ground truth (original image without the mask).

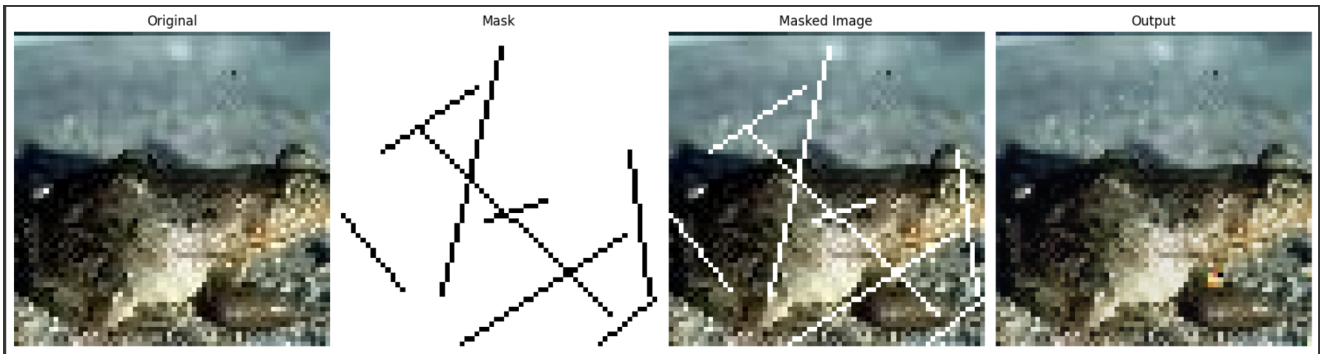
Training the Network
The network is trained on a dataset of images with corresponding masks. The loss function measures the difference between the output image and the

ground truth. Through gradient descent and backpropagation, the network learns to:

- Inpaint the missing areas coherently based on the surrounding context.
- Update the masks to gradually cover the inpainted regions.

3.3.3 Results Using Partial Convolutions:

Utilizing the Tiny ImageNet-200 dataset, our model for image inpainting significantly outperforms traditional techniques like the Fast Marching Method (FMM) in restoring images with extensive damage. By applying partial convolutions, the model selectively processes undamaged pixels, ignoring masked areas to reconstruct missing parts with high fidelity. Although our current model is not exceedingly deep, implementing deeper network architectures could further enhance its performance. Deeper models can potentially provide a more profound contextual understanding, thereby improving the algorithm's efficiency in inpainting images with even larger occluded regions. Such advancements could expedite the inpainting process while ensuring more accurate and seamless restorations.



4.Conclusion and Future Work:

4.1 Conclusion:

In conclusion, our exploration of image inpainting has traversed from the traditional Fast Marching Method (FMM) to the more advanced deep learning-based technique of partial convolutions. FMM, while foundational, is limited by its reliance on the diffusion of information from the image boundary inward, often leading to less accurate restorations when dealing with complex patterns or larger occluded areas. Partial convolutions, on the other hand, leverage the power of deep learning to intelligently reconstruct missing portions of an image.

This method selectively processes only the valid pixels during convolution, allowing the model to focus on existing information and use it to predict the missing parts. The approach has proven to be particularly adept at handling images with significant portions removed, surpassing traditional methods in both the quality of the reconstruction and the ability to maintain the coherence of complex textures and structures within the image.

4.2 Future Work:

Our future work will harness Generative Adversarial Networks (GANs) to elevate image inpainting, especially for images with extensive damage. GANs, with their innovative generative-discriminative interplay, hold the potential to craft detailed, context-aware fill-ins. We'll focus on perfecting GANs for complex areas, enhancing texture blending and edge coherence where partial convolutions fall short.

We plan to tailor GANs for precise detail recovery in intricate or semantically significant regions. Our goal is a system that delivers seamless, undetectable inpainting across any image, which could revolutionize fields from art restoration to medical imaging. Integrating GANs represents a leap toward indistinguishable, realistic image reconstruction, propelling automated image restoration forward.

5.Resources

- https://youtube.com/playlist?list=PL0-GT3co4r2y2YErbmuJw2L5tW4Ew2O5B :
  Linear Algebra Playlist

- https://www.youtube.com/playlist?list=PLS1QulWo1RIa7D1O6skqDQ-JZ1GGHKK-K :OpenCv Playlist

- https://www.coursera.org/specializations/deep-learning : Andrew NG Courses on Machine Learning

- https://www.researchgate.net/publication/238183352_An_Image_Inpainting_Technique_Based_on_the_Fast_Marching_Method :Research Paper for FMM Implementation.

- https://doi.org/10.48550/arXiv.1804.07723 :Research paper referred for Partial Convolutions Implementation

- https://www.kaggle.com/datasets/akash2sharma/tiny-imagenet : LInk to the Imagenet Dataset

- https://github.com/ayulockin/deepimageinpainting.git :Reference taken from this repository