FACE
UNDER-MASK
RECONSTRUCTION

A Graduation Project Dissertation

Prepared By

The Team Members

ID	Name
- 201900547	- غيداء الطاهر احمد محمد
- 201900573	- كاترين حبيب جورج ناداب
- 201900568	- فريدة أحمد توكو سعيد
- 201900554	- فاروق ابراهيم فاروق عبد اللطيف
- 201900534	- عمر نبيل صابر ندا

Supervised By

Dr.

Wessam El-Behaidy

COMPUTER SCIENCE DEPARTMENT

FACULTY OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE

HELWAN UNIVERSITY

June_2023

Document Overview

This document will cover all the information needed to implement our project. In Chapter One, we outline the problem and the idea behind our project and describe the various technologies used to build the model. In Chapter Two, we mention all the related work to our project. In Chapter Three, we explain all the techniques we'll utilize to build our model. In Chapter Four, we cover our custom dataset. In Chapter Five and Chapter Six, we present our model and the implementation details.

List Of Contents

Page

Chapter 1

Introduction

1.1. ABSTRACT

This paper features a method for reconstructing the face under mask but not just a mask it detects and reconstructs the face under 19 different occlusions. We show the performance of our model and the effectiveness of each network separately. Our model starts by detecting the occlusion that is on the face and gradually removes it using a GAN based network which consists of a generator (vanilla convolution and a Partial convolution) and a discriminator to determine how close to a real human our model can predict a face. We showed that the model can detect a variety of occlusions that differ a lot. after detecting, our model will reconstruct the face considering different races and ages male and female of face images. our model is evaluated using a custom dataset. Experimental results confirm our model's effectiveness.

1.2. INTRODUCTION

Throughout the era of the pandemic of COVID-19, we've been dealing with different circumstances that had eventually lead us to innovations that we have been needing to do in the fields of computer vision and deep learning. Wearing masks had become a legitimate subject which made it even harder for detection systems to recognize people and their faces. Many researches were recently made concerning this topic and many were successful and became a hot topic that uses the most recent deep learning models involving image inpainting, attention mechanism, ...etc. To us the problem wasn't about wearing a mask as it was about face occlusions in general.

So, in this project we're aiming to build a model that can reconstruct a face that has been occluded by any of a collection of objects that we found mostly occluding a face such as (masks, sunglasses, scarves, ...etc.). Our aim is to reconstruct the occluded face image in a way to make it as close and real as possible to the true non-occluded face image.

Our model is sub-divided into three main stages:

- _ First, an object detection model combined with a segmentation model.
- _ Second, the image completion (reconstruction) model.
- _ Third, a refine network and a parsing network included in the completion model which leads to more realistic and accurate results.

We will discuss each of these models, their structure and mechanism below in *Chapter 3*.

1.3. MOTIVATION

In the past few years people used to wear a face mask more than ever, which also was not by choice, due to that it became a difficult task to recognize who is the person behind the mask, so our project with the help of all the related work could make that task easier using different techniques and methods to reconstruct the face beneath the mask so the person could be recognized.

1.4. PROBLEM DEFINITION

Most papers used a face mask as an occlusion. The problem we're trying to solve is to make it more general so it could not only reconstruct the face under the mask but to reconstruct the face under any occlusion that could cover any part of the face. adapting this project idea and applying it could help a lot in different fields which we'll be discussing in the future work.

1.5. SIMILAR SYSTEM INFORMATION (REVIEW ON THE PAPERS THAT'S USED IN THE MODEL)

Effective Removal of User-Selected Foreground Object from Facial Images Using a Novel GAN-Based Network is a paper published in 2020 [3], which is the basis of our model.

It's GAN-based model consist of two stages :

1- an object detection module.
2- an image completion module.

Object detection is based on U-net architecture, where the completion module includes two encoders vanilla and partial convolution, which are the main part of this module.

Generative Face Completion is a paper published in 2017 [1], the module is a GAN based but what got us inspired is the Parsing Network they used for regularization which we'll be using in our model.

A Deep Learning Framework to Reconstruct Face under Mask is a paper published in 2022[2], what distinguishes their work is that they used a gender classifier. The module is also GAN based with a modified Mask R-CNN for binary segmentation.

Chapter 2

Related Work

2.1. GENERATIVE FACE COMPLETION AND FACE PARSING

Face completion has been a challenging problem in the field of computer vision ,different papers showed different results and techniques to get a realistic result ,one of the papers (a Generative Face Completion) was published in 2017 .

The paper used the GAN techniques to complete the missed part of the face after removing the mask while making sure that the completed parts are realistic and coherent to the whole face

The algorithm that was used depends on a neural network where the model is trained with a combination of a reconstruction loss, two adversarial losses and a semantic parsing loss to ensure pixel faithfulness and a global contents consistency.

The paper shows that they were able to generate realistic face completion results, the proposed algorithm in the paper is for object completion, given a masked image the goal is to reconstruct the missing contents (masked regions) while keeping the consistency with the whole object and at the same time giving a realistic result.

The proposed model can successfully synthesize semantically plausible contents for the missing facial key parts from random noise which is also flexible to handle variation of masking positions, sizes, or shapes. Finally, they mentioned that the model could be improved with more careful hyperparameter tuning, this paper inspired us to use a parsing network in our model [1].

2.2. A DEEP LEARNING FRAMEWORK TO RECONSTRUCT FACE UNDER MASK REVIEW

Deep learning-based image reconstruction methods showed great success in removing objects from images yet to achieve acceptable results for attributing consistency to gender, ethnicity, expression, and other characteristics is challenging.

In previous work it was hard for the network to distinguish between male and a female face so the generated face for a female could have a male feature and vice versa. To overcome this problem, they proposed a **gender classifier.**

They split the problem into three phases :1- landmark detection 2-object detection for the targeted mask area 3- inpainting the addressed mask region

For object recognition a Faster R-CNN is utilized, the semantic segmentation is performed using a fully convolutional network (FCN). The Resnet50 is used to extract features from images using Mask R-CNN; its aim is to extract low level and high-level information. A Region Proposal Network predicts if an item is present in a given area or not. We were inspired by the object detection module in this paper and that's what we used in our model.

They created a synthetic dataset using the Flicker-Faces-HQ Dataset (FFHQ).the dataset includes 52000 high quality PNG pictures with a size of 512*512 pixels with a huge diversity in age and ethnicity .

The metric results using FFHQ dataset were:

For Male, PSNR (28.82) SSIM (0.93) , Female -> PSNR (30.00) SSIM (0.95)

Using CelebA dataset:

Male-> PSNR (31.40) SSIM (0.97) Female PSNR (33.32) SSIM (0.98)

The higher the value of the metric, the more efficient it will be, and in the case of SSIM, the closer to 1, the better the quality image will be [2].

2.3. VANILLA AND PARTIAL CONVOLUTION MODEL

In this study [3], a novel and user-friendly method for face de-occlusion in facial images is proposed. The user can select the object to be excluded, with an emphasis on commonly occurring occluding things like (hands, a medical mask, microphone, sunglasses, and eyeglasses).

To support the removal of the mask from the face, we will concentrate on the key components of this method in this review. In order to achieve well-integrated predictions, this paper conducted extensive research on earlier papers that addressed the same subject. They discovered that most of the earlier methods relied solely on vanilla convolution as the foundation of their deep-learning-based networks. However, this method produces severe visual artifacts, particularly at the boundaries of the valid and affected regions of the image. So, they provide an unique GAN-based with two discriminators to address the restrictions. An object detection module and an image completion module make up the model's two stages. Additionally, they incorporate both vanilla and partial convolution in parallel . These two encoders are used in parallel to provide content that is well-integrated, crisp, and full of minute details.

It uses separate encoders for the input image and the target object label, unlike U-Net, which only has one encoder. if they use a single encoder for both the input image and the target object label, it will only encode the information in the input image and ignore the information in the target object label. The discriminator in this case classifies the label of the considered target object rather than using a binary classifier (real or fake).

Generator, discriminators, and perceptual network are the main building parts of the completion module. Two encoders, partial convolution and vanilla convolution, make up the generator. They are both components of the U-Net architecture. The vanilla encoder employs a convolution network and take as input an image and a binary segmentation map. as previously said only utilizing it does not produce good results. for this reason, they also use a partial convolution encoder which take the inverse segmentation map and the input picture . as u-net Instead than using standard convolution layers, it uses partial convolution layers . Since it only applies the convolution operation to valid regions, the segmentation map gets thinner after each convolution operation, which increasing valid regions. They use atrous convolution between the output of two encoders and decoder to integrating the face's remaining features with the created missing section. The discriminator Dglobal p examines the full-sized produced image while Dlocal focuses on the generated missing region further.

Because there isn't a dataset that includes both occlusion objects and face images, they created one using the CelebFaces Attributes Dataset (CelebA) and CelebA-HQ dataset.

The quantitative performance using four matrics : 1) Structural SIMilarity (SSIM) , 2) Peak Signal to Noise Ratio (PSNR), 3) Naturalness Image Quality Evaluator (NIQE) , and 4) Blind/Referenceless Image Spatial Quality Evaluator (BRISQUE) .

The advantage of this paper is that it removes complex things like masks that cover a substantial portion of the face and uses vanilla and partial convolution in tandem to produce results that are visually pleasant and outperform other models. disadvantage that failure situations happen more often when the hands are involved, especially when more than 70% of the face is obscured. [3]

2.4. ATTENTION MODELS

2.4.1. Toward High-quality Face-Mask Occluded Restoration

The most challenging part of this problem is that there is a lot of images with a lot of visual variance of masks in the real world so that they collect a large dataset to cover this visual variance and they use a lot of datasets to make there one using AR, CelebA, CAS-PEAL, CMU-PIE, MAFA, and RMFRD datasets and follows coarse-to-fine architecture for face restoration

full convolution neural network with convolutions and up-sampling structure was used to detect the face-mask region

Applying a U-Net architecture in coarse net to generate a coarse image and for refine network they use a tow-stream enc using self-adaptive contextual attention and dilated convolution to generate a refined image.

Model & Techniques: CNN for face-mask region detector, Course network, refine network (tow-stream network one of them have self-adaptive contextual attention).

Datasets used: AR, CelebA, CAS-PEAL, CMU-PIE, MAFA, and RMFRD.

Advantages:

1- they use to collect a lot of datasets which makes it possible for their model to solve the challenging problems.

2- the idea of using self-adaptive contextual attention to use the feature information from background patches to generate masked patches helped whit perceptual loss.

Disadvantages: this model fails to generate an accurate image of side face images and blurred images if there is a masked face with glasses.

Evaluation Metrics used: Structural Similarity (SSIM) = 0.864, Peak Signal to Noise Ratio (PSNR) = 26.19bB. [4]

2.4.2. Contrastive Attention Network with Dense Field Estimation for Face Completion

This paper proposes a self-supervised Siamese inference network based on contrastive learning for face completion. They assumed that two identical images with different masks form a positive pair while a negative pair consists of two different images. Contrastive learning aims to maximize (minimize) the similarities of positive pairs (negative pairs) in representations. To deal with geometric variations of face images, they integrated a dense correspondence field into their network, this field binds 2D and 3D surface spaces which can preserve the expression and pose of the input. They further proposed a multi-scale decoder with a novel dual attention fusion module (DAF) that can explore feature interdependencies in spatial and channel dimensions and blend features in missing regions and known regions naturally. This multi-scale architecture was beneficial for the decoder to utilize discriminative representations learned from encoders into images.

Training their method had two stages. In the first stage, the inference network is trained through contrastive learning until convergence. And in the next stage, the pre-trained encoder and the decoder are jointly trained with the fusion module. For synthesizing richer texture details and correct semantics, loss functions were used such as the element-wise reconstruction loss and many others. In addition, they employed the identity preserving loss function to ensure that the identity information of the generated images remains unchanged.

They conducted their experiments for two main tasks face completion and face verification. Their model was trained and tested many times on different publicly available datasets. They used CelebA, CelebA-HQ, FFHQ, and L2SFO for face completion training and testing. As for face verification, they used the training sets of CelebA and Multi-PIE and the test sets of LFW and IJB-C. Their method was implemented by the Pytorch framework and trained on four NVIDIA TITAN Xp GPUs (12GB). Peak signal-to-noise ratio (PSNR), structural similarity index (SSIM), and Fréchet Inception Distance (FID) were used as evaluation metrics for most testing. Their method achieved the best quantitative results in three metrics on all testing sets of CelebA (PSNR: 33.26, SSIM: 0.9769, FID: 0.7981), CelebA-HQ, and FFHQ compared with other state-of-the-art methods, which consists of two image inpainting methods, GMCNN and DFNet, and three image-to-image translation methods: Spade, CycleGAN and CUT. They then conducted experiments for face completion on a real-world masked face dataset (RMFD) and got an F1-Score of 0.0493. The face verification tests on LFW and IJB-C datasets achieved high accuracies as well while using the area under the ROC curve (AUC) as the evaluation metrics in the experiments.

Advantages: Using the self-supervised Siamese inference network helped improve the generalization and robustness of encoders. Together with the dense correspondence field and the novel dual attention fusion module (DAF) clearly showed in their detailed and extensive conducted experiments that the proposed approach not only achieves more appealing results compared with state-of-the-art methods but also improves the performance of masked face recognition dramatically.

Disadvantages: Really heavy and quite expensive process. The method consumes a lot of hardware and memory capacity. [5]

2.5. RESIDUAL ENCODER DECODER NETWORK AND ADAPTIVE PRIOR FOR FACE PARSING

The paper proposed a novel pixel wise face parsing method called residual encoder decoder network (RED Network) they combine feature rich encoder decoder framework with adaptive prior mechanism the encoder and the decoder framework extract the features using Resnet after that features will be decoded. The system consists of three parts:

1) residual encoder network, consist of series of residual encoder units which extract features of facial parts

2) residual decoder network consists of series of residual decoder unit, which stores the feature map to original resolution to obtain a pixel wise score map

3) adaptive prior which refines the score map

They mentioned that the RED Network was pursuing higher accurate face parsing. To overcome the appearance ambiguity between facial parts; an adaptive prior mechanism was used in terms of the decoder prediction confidence which allows refining the results.

They mentioned that the accuracy was 97 and the F-measure increased to 0.90, this is using Helan and LFW dataset [6]

2.6. GAN-BASED FACE MASK REMOVAL USING FACIAL LANDMARKS AND PIXEL ERRORS IN MASKED REGION

This paper proposes to use generative adversarial networks (GAN) to complete the facial region that hidden by the mask. which is an important cue for communication facilitation, in human face image.

First, the paper mentioned related works: Inpainting which is a technique used to repair scratches and holes in a part of an image, (Xie et al., 2012) by using a denoising auto encoder, and DeepFill (Yu et al., 2018; Yu et al., 2019), generative adversarial networks . On these methods when there is a huge hole concentrated in one place in the image, such as mask, they may fail.

So, they propose to define custom loss functions that focus on the errors of the feature point coordinates of the face and the pixels in the masked region. By using pix2pix as a baseline which is a GAN that learns the correspondence between a pair of images and given one image, generates the other corresponding image. Next, they add two new custom terms to the loss function formula of the generator, the mean squared error of the pixel values limited to the masked region, and the mean of the error of the feature point coordinates predicted from the face contour. Using dlib library to obtain the feature point coordinates

The dataset was created by them using MaskTheFace to put the mask on the real faces images .1400 images from Flicker-Faces-HQ (FFHQ) Dataset ,588 from the Karolinska Directed Emotional Faces (KDEF) Dataset, 343 from UTKFace Dataset.

For quantitative metrics they use mean squared error (MSE), peak signal-to-noise ratio (PSNR), structural similarity (SSIM), and learned perceptual image patch similarity (LPIPS). after do training for 300 epochs, SSIM with **0.926** and LPIPS with **0.0459** give the best values . they also use human-rated quality score (HQS) gives **4.78±1.08** on the qualitative evaluation experiment, and accuracy **94.00±5.16**.

The advantage in this paper was that they designed facial expression identification with range of accuracy between (75-100)% for different expressions.

The disadvantage is the result may get worse if the given face image is wearing a mask different from the one used in the training dataset. [7]

2.7. MASK-RCNN AND U-NET ENSEMBLED FOR NUCLEI SEGMENTATION

U-Net and Mask-RCNN are used widely in object segmentation. Each of them has different strengths and failures as shown in this paper.

U-Net is better than the mask R-CNN in image segmentation due to its high performance in feature extraction but can't detect objects well and the Mask R-CNN is better in object detection so they made an Ensemble model to combine the predictions of both models, where structural properties of the prediction mask were used as features and intersection over union (IoU) with ground truth as target [8]

Chapter 3

Background

3.1. OBJECT DETECTION AND SEGMENTATION

3.1.1. Object Detection and Instance Mask Segmentation Module

The module produces the binary segmentation map, mask object represented by 1 and the rest by 0. The network is based on the Mask R-CNN which is a version of Faster R-CNN that's used to classify and identify different objects in an image.

Mask R-CNN generates an object mask for each item that's detected, each having a class and a bounding box.

For object recognition a Faster R-CNN is utilized, the semantic segmentation is performed using a fully convolutional network (FCN). The Resnet50 is used to extract features from images using Mask R-CNN; its aim is to extract low level and high-level information. A Region Proposal Network predicts if an item is present in a given area or not [2].

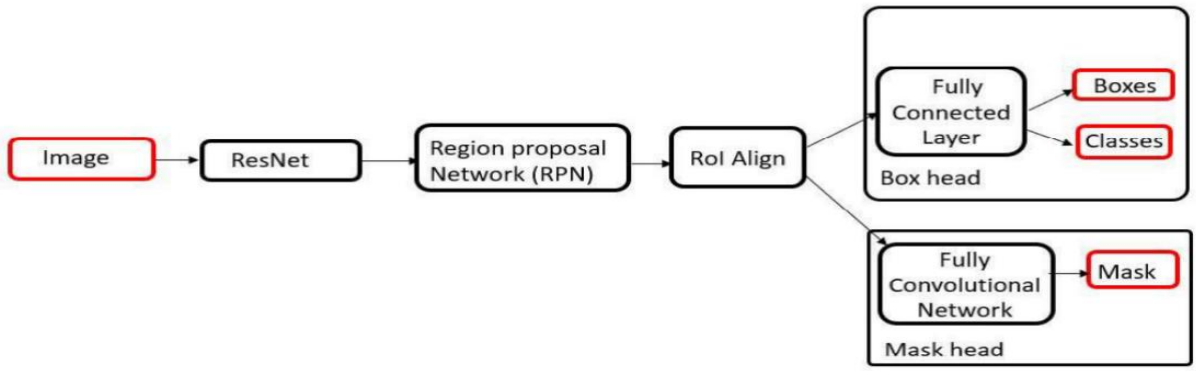


Figure 3.1: *The Mask R-CNN workflow*

Masked R-CNN Object detection: The U-Net is better than the mask R-CNN in image segmentation due to its high performance in feature extraction but can't detect objects well; the mask R-CNN is better at performing the object detection which is supported by using Resnet50 to extract more features from the image.

The Resnet50 extracts both low level and high-level information from a picture.

The aim of a Region Proposal Network is to predict if an item is present in a given area or not.

Mask R-CNN can also estimate human poses as well as instance segmentation and bounding box object detection

Instance Segmentation: The additional part on Faster R-CNN is a branch for predicting segmentation masks(FCN) on each Region of Interest (RoI) which predict a segmentation mask in a pixel-to-pixel manner, that's in parallel with classification and bounding box [2].

3.2. IMAGE INPAINTING

3.2.1. U-NET

Its network features a u-shaped design, and it was the winning entry in the 2015 ISBI cell tracking challenge [12]. They describe a network and training technique that heavily relies on data augmentation to function more effectively. The architecture comprises of a symmetric expanding path that permits exact localization and a contracting path to capture context. They also demonstrate how a network of this kind may be trained entirely from a small number of images.

They are based on a more sophisticated design known as a "fully convolutional network." The fundamental idea is to add additional layers on top of a typical contracting network, replacing the pooling operators with upsampling operators. As a result, the output's resolution is increased by these layers. In order to localize, the network propagates context information to higher resolution layers by combining high resolution features from the contracting path with the upsampled output. one important thing that the network does not have any fully connected layers and only uses the valid part of each convolution which mean the segmentation map only contains the pixels, for which the full context is available in the input image.

This network's architecture consists of a contracting path (left side), also known as an encoder, and an expansive path, also known as a decoder. The contracting path follows the typical architecture of a convolutional network using a convolutional network repeated twice with a 3x3 convolution after each of them (ReLU) and , 2x2 max pooling operation with stride 2. When the input size was decreased, the number of feature channels doubled. in an extended path using 2x2 convolution, which reduces the number of feature channels in half. Cropping, which is the concatenation of a feature map from a contracted path to a contracted path, is one item that is required. due to the loss of boundary pixels in each convolution. The network consists of 23 convolutional layers in total.

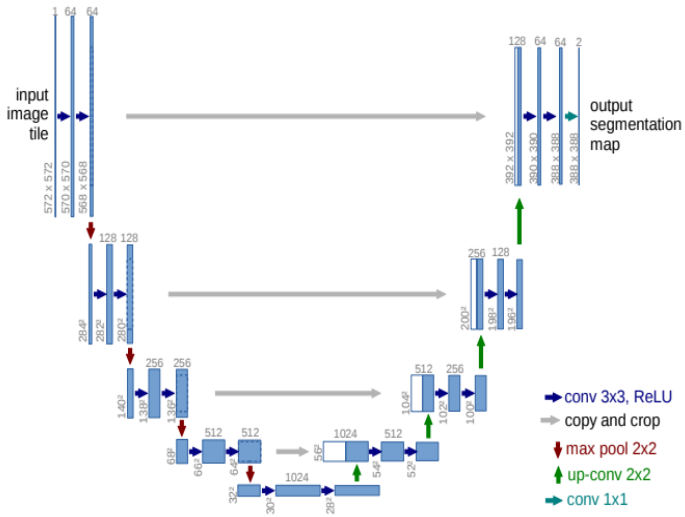


Figure 3.2: *U-net architecture*

3.2.2. Partial Convolution

It is where the convolution is conditioned on just valid pixels after being masked and renormalized. also Include a mechanism to automatically generate an updated mask for the next layer as part of the forward pass.

Partial Convolutional Layers, which refers to partial convolution operation and mask update function, propose to handle irregular masks by first performing a masked and re-normalized convolution operation and then performing a mask update step. The convolutional results depend only on the non-hole regions at every layer , where mask-update removes any masking so even the biggest masked holes will eventually disappear because of subsequent updates, leaving only valid responses in the feature map.

Many earlier works had trouble getting decent inpainting results using skip links in a U-Net with typical convolutions, thus in [3] they designed a u-net network that replaced convolutional layers with partial convolutions and mask updates to provide state-of-the-art inpainting results.

To explain Partial Convolutional Layer , Let W be the convolution filter weights for the convolution filter and b is the corresponding bias. X are the feature values (pixels values) for the current convolution (sliding) window and M is the corresponding binary mask. The partial convolution at every location, is expressed as:

$$x' \quad \begin{cases} \mathbf{W}^T(\mathbf{X} \odot \mathbf{M}) \frac{\mathrm{sum}(\mathbf{1})}{\mathrm{sum}(\mathbf{M})} + b, & \text{if } \mathrm{sum}(\mathbf{M}) > 0 \\ 0, & \text{otherwise} \end{cases}$$

where $\odot$ denotes element-wise multiplication, and 1 has same shape as M but with all elements being 1. As can be seen, output values depend only on the unmasked inputs. The scaling factor sum(1)/sum(M) applies appropriate scaling to adjust for the varying amount of valid (unmasked) inputs.

After each partial convolution operation, we then update our mask as follows: if the convolution was able to condition its output on at least one valid input value, then we mark that location to be valid. This is expressed as:

$$m' = \begin{cases} 1, & \text{if } \mathrm{sum}(\mathbf{M}) > 0 \\ 0, & \text{otherwise} \end{cases}$$

After enough partial convolution layer applications any mask will eventually be all ones if the input includes any valid pixels. [9]

3.2.3. GAN

A generative adversarial Network (GAN) is a Machine Learning framework. Actual working using GAN started in 2017 with human faces to adopt image enhancement to be able to produce better illustration at high intensity.
It uses two different networks the first is called a generator that tries to generate fake images and the second one is the discriminator that classifies the generated images to fake or real
If at any point the discriminator cannot distinguish between a generated image and an actual image it's considered as converged.

In the training phase it trains to learn to produce novel information that is close to the training set.
A generative model G captures the data distribution and a discriminative model D that estimates the probability that the image is an actual representation rather than a generated image by the generator G.
The training procedure of G aims to maximize the probability of D making a mistake. This framework corresponds to a minimax two player game. [10]

3.2.4. Convolutional Neural Network

There are numerous variants of CNN architectures in the literature. However, their basic components are very similar. Taking the famous LeNet-5 as an example, it consists of three types of layers, namely convolutional, pooling, and fully connected layers. The convolutional layer aims to learn feature representations of the inputs. Convolution layer is composed of several convolution kernels which are used to compute different feature maps. Specifically, each neuron of a feature map is connected to a region of neighboring neurons in the previous layer. Such a neighborhood is referred to as the neuron's receptive field in the previous layer.

The new feature map can be obtained by first convolving the input with a learned kernel and then applying an element-wise nonlinear activation function on the convolved results. Note that, to generate each feature map, the kernel is shared by all spatial locations of the input. The complete feature maps are obtained by using several different kernels.
The activation function introduces nonlinearities to CNN, which are desirable for multi-layer networks to detect nonlinear features.

The pooling layer aims to achieve shift-invariance by reducing the resolution of the feature maps. It is usually placed between two convolutional layers. Each feature map of a pooling layer is connected to its corresponding feature map of the preceding convolutional layer.

After several convolutional and pooling layers, there may be one or more fully connected layers which aim to perform high-level reasoning. They take all neurons in the previous layer and connect them to every single neuron of the current layer to generate global semantic information.

Note that a fully connected layer is not always necessary as it can be replaced by a 1×1 convolution layer. The last layer of CNNs is an output layer. For classification tasks, the SoftMax operator is commonly used [8]. Another commonly used method is SVM, which can be combined with CNN features to solve different classification tasks.
The optimum parameters for a specific task can be obtained by minimizing an appropriate loss function defined on that task [11].

Chapter 4

Custom Dataset

4.1. CREATING A CUSTOM DATASET

For the work involved with this project, we had to find a dataset that contains facial image pairs with and without occlusions to train our model in a supervised manner; similar projects only involved masks as occlusions in their cases. Most papers that work with more than one occlusion item had to create their dataset on their own, so all the datasets that we found during our research were focused only on face mask occlusions knowing that in recent times, the COVID 19 was the main topic that people had to interact and live with, so many of these similar projects were most concerned with reconstructing a masked face to help with security cameras and other systems that were mainly dealing with face detection and recognition. In this project, we aim to rebuild an occluded detected face with more than one occlusion item taken into consideration; for that, we will deal with face data images and occlude them with some occlusions of our choice.

4.1.1. Gathering the Data (face images)

So, to start with gathering the face images, many related datasets already exist and are open source, i.e., they are free to public use. We gathered our data images by combining two well-known datasets a 30,000 images CelebA-HQ dataset and a 23,708 UTKFace dataset,

CelebA-HQ: was Introduced by Karras et al. in Progressive Growing of GANs for Improved Quality, Stability, and Variation [13]

The CelebA-HQ dataset is a high-quality version of CelebA dataset that consists of 30,000 images at 1024×1024 resolution.

UTKFace: was Introduced by Zhang et al. in Age Progression/Regression by Conditional Adversarial Autoencoder [14].

The UTKFace dataset is a large-scale face dataset with long age span (range from 0 to 116 years old). The dataset consists of over 20,000 face images with annotations of age, gender, and ethnicity. The images cover large variation in pose, facial expression, illumination, occlusion, resolution, etc. This dataset could be used on a variety of tasks, e.g., face detection, age estimation, age progression/regression, landmark localization, etc.

We found that most papers were concerned with a specific race. Their models weren't performing well on other different races. We chose to combine these datasets because we wanted our data to contain a variety of men's and women's faces from multiple races and different ages so that our model would be more aware of these broad changes when dealing with human faces.
We combined images from both datasets to have a some of 53,708 face images to work with.

4.1.2. Preprocessing Our Images (two rounds)

We preprocessed our data 2 times to ensure that the data was clean and ready for the training uses.

Round 1 *(before occlusions and segmentation):*

We needed images with clear faces to train our model for the inpainting process.
The images in the CelebA-HQ dataset had wider dimensions; they had unnecessary details and backgrounds that could distract the model. In the UTKFace dataset, images were the size of 200x200 with only faces in them. We extracted the face parts from the CelebA-HQ images and resized them into 200x200 images with the default pre-trained opencv Haarcascade frontalface extraction model [15].
Some contained occluded faces, and some had baby faces; these images aren't relevant to our case. So, we had to remove them manually, which left us with 45,597 out of the 53,708 we gathered at first. Each is labeled from 1 to 45,597. Then we divided them into females and males, with 27,426 females and 18,171 males.

Round 2 *(after occlusions and segmentation):*

After we occluded the images and segmented them, some occluded images appeared to be missed from the first round and some others were interfering with the occlusions and segmentations weren't clear and we had to remove them as well.

Total number of images left out of both rounds : 39,689 images ~40,000 images.

4.1.3. Occluding The Images

We gathered around 19 different occlusion categories as transparent PNG files to occlude our images, each with over 20 different types and shapes. Using the Droplet functionality in Photoshop, we could occlude all of the 39,689 images, occluding the images manually with different (occlusion) sizes and positions. Each image has only one object as occlusion.
The categories we chose and found to be generally occluding human faces are: Books, Boxes, Candies, Caps, Cigarettes, Cups, Envelopes, Flowers, Food(fast food), Fruits, Gloves, Hands, Keys, Masks, Mics, Pens, Phones, Scarves, and Sunglasses.

***Figure** 4.1 Showing samples of the occlusions we used.*

4.1.4. From Occlusions to Segmentations

A Variety of occlusions with corresponding object segmentations:

After occluding our images with the object varieties we had, we needed to define each binary mask segmentation corresponding to each occlusion in the occluded images; we used a bitwise difference method to get the difference between the occluded and the original images, i.e., the occluding object, and save it as a binary mask. We labeled the segmentations with the same label names of the images containing their corresponding occlusion. Segmenting the occluded images was essential for upcoming stages and training of our modules.
We used OpenCV the Open Computer Vision Library [16] and Pillow the Python Imaging Library [17] to create these bitwise difference segmentation masks.

```
def get_difference(imageE, image):

    # compute difference

    difference1 = cv2.subtract(image, imageE)
    difference2 = cv2.subtract(imageE, image)

    difference1 = toImgPIL(difference1)
    difference2 = toImgPIL(difference2)

    imageE = toImgPIL(imageE)
    image = toImgPIL(image)

    difference3 = ImageChops.difference(image, imageE)

    segmentation1 = addTowImages(ImageChops.invert(
        difference1), ImageChops.invert(difference2))
    segmentation1 = ImageChops.invert(segmentation1)

    segmentation1 = ImageChops.multiply(segmentation1, difference3)

    segmentation1 = convertTobinary(segmentation1)
    segmentation2 = convertTobinary(difference3).convert(
        "RGB").filter(ImageFilter.MinFilter(3))

    segmentation1 = deNoising(segmentation1.convert("RGB"))

    segmentation1 = convertTobinary(segmentation1).convert("RGB")

    segmentation = addTowImages(segmentation1, segmentation2).convert("1")

    return segmentation
```

***Figure** 4.2 Showing samples of the occluded images with their corresponding binary object segmentation masks and the bitwise difference function we used.*

4.2. CREATING TWO VERSIONS OF THE DATASET

After we've finished gathering, segmenting, and preprocessing the images, we used them to create two different versions of the dataset, each with a specific structure to meet the training needs of each module in our model. So, we can train our modules separately and visualize their performance before combining them. We used the same images and their corresponding segmentations to create both versions, the object detection dataset and the image inpainting dataset named One Dataset and MFDataset, respectively. Both datasets contain only a portion of the images we gathered before; we used only 21,931 out of the 39,689 preprocessed images for our training and testing on Google Colaboratory.

4.2.1. Object Detection Dataset (One Dataset)

The object detection dataset focused on providing data for detecting the occlusions from the occluded images. We only needed occluded files for this task.

The One Dataset is with 90% to 10% train-test split ratio with 19,745 training images and 2,186 testing images. Images we used in our experiments corresponded to only 12 occlusion classes out of the 19 categories. We used the occluded files and their corresponding predefined segmentations for our training and testing experiments, we didn't need the original ones for the detection task, so we excluded them. Also, male-female classification wasn't necessary, so we put them all in one directory.

The 12 Categories included are Boxes, Candies, Caps, Cigarettes, Cups, Envelopes, Food(fast food), Flowers, Fruits, Gloves, Hands, and Masks.

Each class includes an average of 1,500 to 1,700 train images and 180 to 190 test images. Segmentation files are labeled with the corresponding image name, the class name, and the instance count, as shown in *Figure 4.3*. Since they are all single instances, all the instance counts are 0. All images were size 200x200 px.

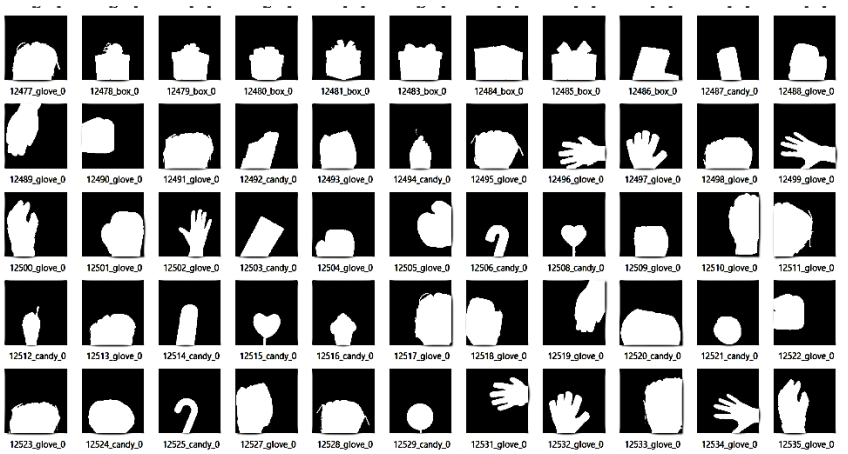


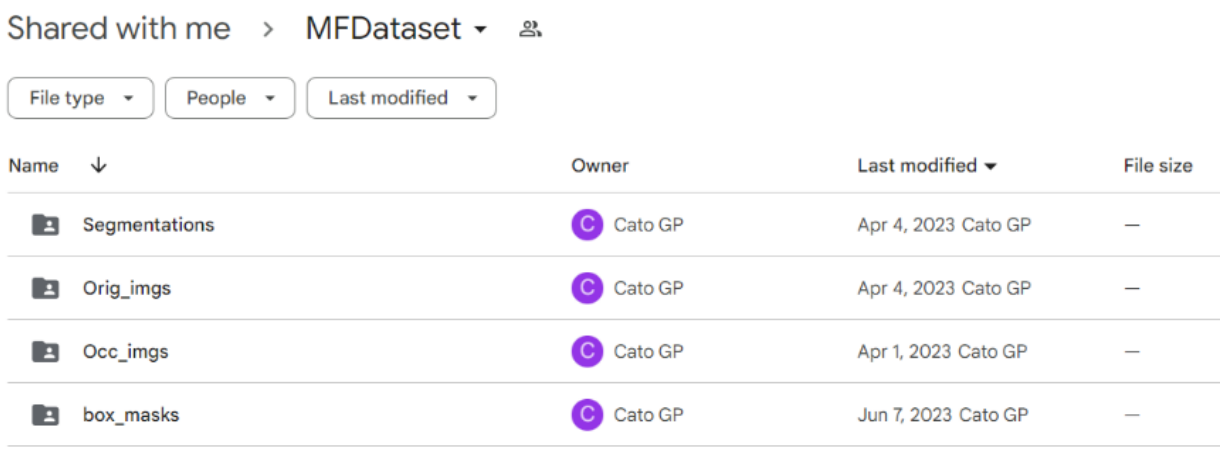
Figure 4.3 *Showing some samples from the One Dataset labeled segmentation files.*

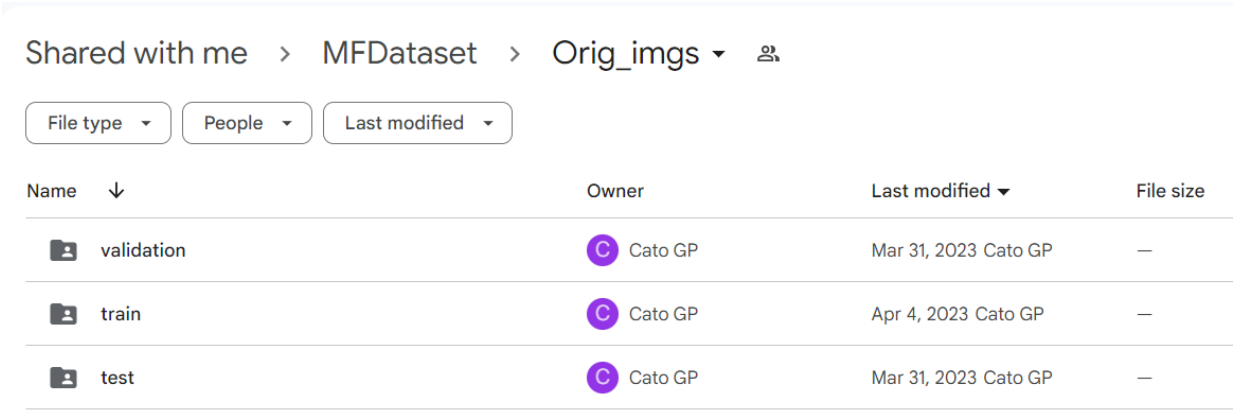
4.2.2. Image Inpainting Dataset (MFDataset)

We created the MFDataset for the tasks of the image inpainting module. It contains all the files, the original images, and the occluded images with the corresponding object segmentations. We classified the images as males and females and divided them into train, validation, and test with ratios of 80% to 10% to 10%, respectively. The MFDataset contains 17,559 train images, 2186 validation images, and 2,186 test images; we labeled all related images with the same names (numbers).

This dataset focused on the image inpainting process of the model; therefore, we also added segmentations of the bounding boxes predicted with the object detection module with the object segmentations to obtain better inpainting performance after detecting the objects.

We relied on this version of our dataset for all the related model training experiments we did on Colaboratory Notebooks provided by Google. As in the One Dataset, all images were size 200x200 px.





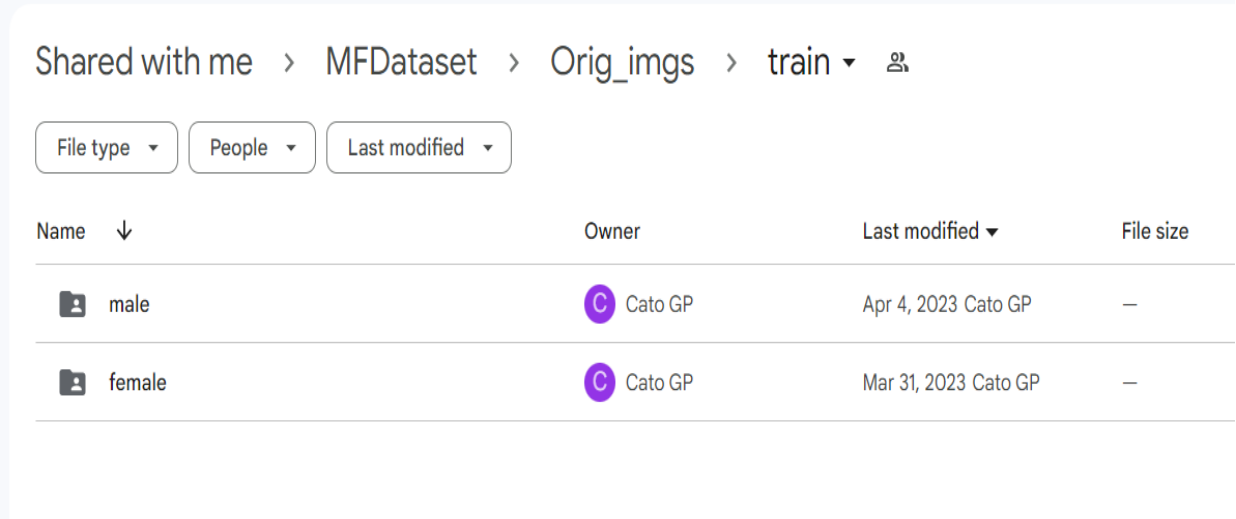


***Figure** 4.4 Showing an example of the file structure of the MFDataset. All the folders have the same file structure except for the box_masks folder, which contains the bounding boxes segmentation files.*

Chapter 5

The Model

5.1. MODEL BLOCK DIAGRAM

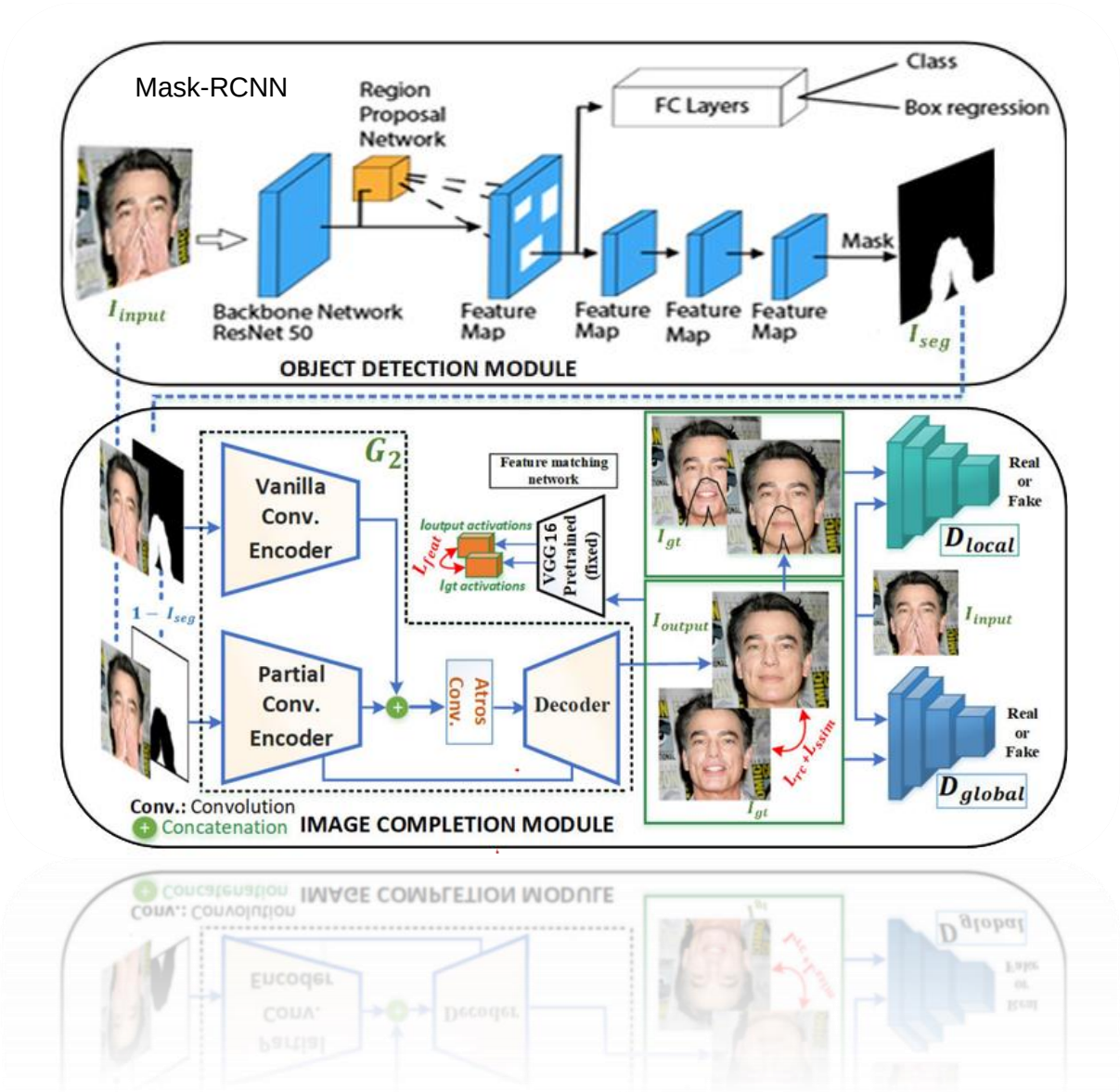


***Figure** 5.1 In our model's block diagram, the architecture is divided into two main modules: Object detection module and Image completion module.*

5.2. THE OBJECT DETECTION MODULE

For the object detection module, we had to train a detector on its own to detect a set of predefined occlusion objects that we mentioned earlier in the dataset, then build a mask for each instance of the detected objects to produce a binary segmentation map for the occluded image that only focuses on the occluding items, for it then to be sent to the inpainting module as an input to build the impaired regions in a face. As we mentioned before, we will be using a Mask-RCNN base architecture together with a Resnet50 backbone to get to a finer mask of the detected objects. We could easily build these structures with the help of Detectron2 [18] library which is an open-source code library that is considered as Facebook AI Research's next generation library that provides state-of-the-art detection and segmentation algorithms. It is the successor of Detectron and mask-rcnn benchmark.

5.2.1. Preprocess The Dataset

Since the provided models were mostly pretrained on COCO dataset, we also had to provide our custom data images with COCO format annotations to be capable of training these models on our data. To do that we had to go through two steps:

1. Build binary object segmentations for our occluded images.
2. Convert the binary segmentations to json annotation files.

Since we occluded the original images manually (each containing only one occlusion), we could build the binary segmentations by subtracting the original images from the occluded ones.

After we had had the segmentations for all occluded images, we converted them to COCO format annotations with the help of another open-source library called Pycococreator [19].

We were then able to train Detectron2 Mask-RCNN pre-trained provided model.

Following an example:

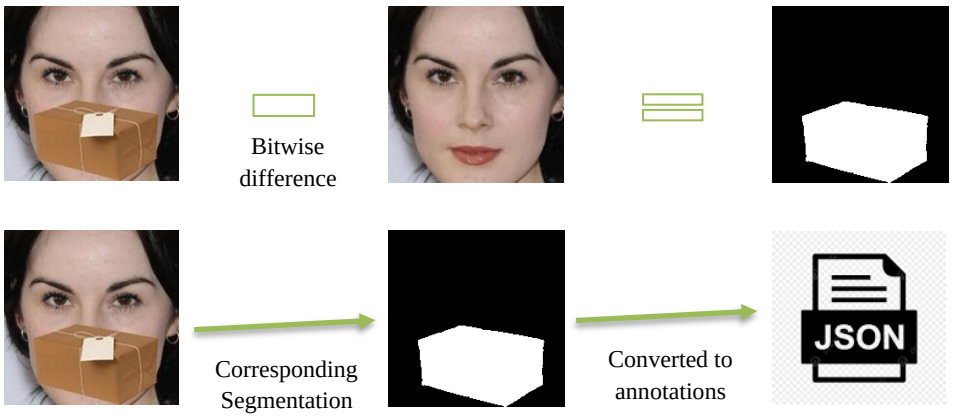


***Figure** 5.2 Get the object segmentation by subtracting the original image from the occluded image, then rename the segmentation image with an image_class_instance name and then convert it to json annotation. Ex: 828_box_0 (image 828, box class, instance 0 (1 instance)).*

5.2.2. Train The Detector

When using Detectron2 library, we first register our custom data with the annotation files as COCO instances, then train the model. We registered our dataset under the name of 'One Dataset'. The dataset had 200x200-sized images. More details concerning the dataset, are mentioned in *Chapter 5*.

COCO Format Dataset

Register

When you use Detectron2, before you actually train the model you need to register it.

```
DATA_SET_NAME = "One Dataset"
ANNOTATIONS_TRAIN_FILE_NAME = "train.json"
ANNOTATIONS_TEST_FILE_NAME = "test.json"
```

```
# TRAIN SET
TRAIN_DATA_SET_NAME = f"{DATA_SET_NAME}-train"
TRAIN_DATA_SET_IMAGES_DIR_PATH = "/content/drive/MyDrive/One Dataset/imgs/train"
TRAIN_DATA_SET_ANN_FILE_PATH = os.path.join("/content/drive/MyDrive/One Dataset/", ANNOTATIONS_TRAIN_FILE_NAME)

register_coco_instances(
    name=TRAIN_DATA_SET_NAME,
    metadata={},
    json_file=TRAIN_DATA_SET_ANN_FILE_PATH,
    image_root=TRAIN_DATA_SET_IMAGES_DIR_PATH
)

# TEST SET
TEST_DATA_SET_NAME = f"{DATA_SET_NAME}-test"
TEST_DATA_SET_IMAGES_DIR_PATH = "/content/drive/MyDrive/One Dataset/imgs/test"
TEST_DATA_SET_ANN_FILE_PATH = os.path.join("/content/drive/MyDrive/One Dataset/", ANNOTATIONS_TEST_FILE_NAME)

register_coco_instances(
    name=TEST_DATA_SET_NAME,
    metadata={},
    json_file=TEST_DATA_SET_ANN_FILE_PATH,
    image_root=TEST_DATA_SET_IMAGES_DIR_PATH
)
```

We can now confirm that our custom dataset was correctly registered using MetadataCatalog.

```
[
    data_set
    for data_set
    in MetadataCatalog.list()
    if data_set.startswith(DATA_SET_NAME)
]
```

```
['One Dataset-train', 'One Dataset-test']
```

***Figure** 5.3 Register our custom dataset to Detectron2 for training.*

We then visualize an input sample with the annotations from the registered dataset :

Visualize

Let's take a look at single entry from our train dataset.

```
metadata = MetadataCatalog.get(TRAIN_DATA_SET_NAME)
dataset_train = DatasetCatalog.get(TRAIN_DATA_SET_NAME)

dataset_entry = dataset_train[1]
image = cv2.imread(dataset_entry["file_name"])

visualizer = Visualizer(
    image[:, :, ::-1],
    metadata=metadata,
    scale=0.8,
    instance_mode=ColorMode.IMAGE_BW
)

out = visualizer.draw_dataset_dict(dataset_entry)
cv2_imshow(out.get_image()[:, :, ::-1])
```

***Figure** 5.4 Visualizing a sample input from the registered data.*

For the training we used the Mask-RCNN model architecture with backbone ResNet50 pre-trained on COCO Dataset, we trained the model on a Colab notebook with Detectron2 library and PyTorch framework using 1TeslaGPU accelerator.

We selected a list of hyperparameters to train the Mask-RCNN Detector model.

Hyperparameters used in training:

Train-test split ratio: 90% to 10% respectively, *Learning rate:* 0.001,

Maximum iteration(epoch): 2000, *Images per batch:* 2,

Mask format: bitmask, *ROI heads batch size per image:* 64,

Number of classes: 19, *Number of workers (dataloaders):* 2,

Architecture: Mask-RCNN and backbone ResNet50 with COCO pretrained weights.

And a list of other parameters which were with default values in the Detectron2 model training.

Then we saved our detector model in a .pth file format named as ‘model_final.pth’.

Train Model Using Custom COCO Format Dataset

Configuration

```
# HYPERPARAMETERS
ARCHITECTURE = "mask_rcnn_R_50_FPN_3x"
CONFIG_FILE_PATH = f"COCO-InstanceSegmentation/{ARCHITECTURE}.yaml"
MAX_ITER = 2000
EVAL_PERIOD = 200
BASE_LR = 0.001
NUM_CLASSES = 19

# OUTPUT DIR
OUTPUT_DIR_PATH = os.path.join(
    DATA_SET_NAME,
    ARCHITECTURE,
    datetime.now().strftime('%Y-%m-%d-%H-%M-%S')
)

os.makedirs(OUTPUT_DIR_PATH, exist_ok=True)
```

```
cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file(CONFIG_FILE_PATH))
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url(CONFIG_FILE_PATH)
cfg.DATASETS.TRAIN = (TRAIN_DATA_SET_NAME,)
cfg.DATASETS.TEST = (TEST_DATA_SET_NAME,)
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 64
cfg.TEST.EVAL_PERIOD = EVAL_PERIOD
cfg.DATALOADER.NUM_WORKERS = 2
cfg.SOLVER.IMS_PER_BATCH = 2
cfg.INPUT.MASK_FORMAT='bitmask'
cfg.SOLVER.BASE_LR = BASE_LR
cfg.SOLVER.MAX_ITER = MAX_ITER
cfg.MODEL.ROI_HEADS.NUM_CLASSES = NUM_CLASSES
cfg.OUTPUT_DIR = OUTPUT_DIR_PATH
```

***Figure** 5.5 Detector model hyperparameters and configuration for training.*

We trained the Mask-RCNN model to detect 12 objects only out of the 19 objects we had, with a total of 19,745 training occluded images and 2,186 testing occluded images.

Training

```
trainer = DefaultTrainer(cfg)
trainer.resume_or_load(resume=False)
trainer.train()
```

```
    )
  )
)
[03/03 11:51:57 d2.data.datasets.coco]: Loaded 19745 images in COCO format from /content/drive/MyDrive/One Dataset/train.json
[03/03 11:51:57 d2.data.build]: Removed 0 images with no usable annotations. 19745 images left.
[03/03 11:51:58 d2.data.build]: Distribution of instances among all 19 categories:
```

category	#instances	category	#instances	category	#instances
book	0	box	2598	candy	2600
cap	2703	cigarette	2253	cup	2060
envelope	2599	fastfood	2609	flower	2602
fruit	2523	glove	2551	hand	2473
keys	0	mask	2559	mic	0
pen	0	phone	0	scarf	0
sunglasses	0				

```
[03/03 11:54:00 d2.data.datasets.coco]: Loaded 2186 images in COCO format from /content/drive/MyDrive/One Dataset/test.json
[03/03 11:54:01 d2.data.build]: Distribution of instances among all 19 categories:
```

category	#instances	category	#instances	category	#instances
book	0	box	186	candy	187
cap	190	cigarette	165	cup	153
envelope	188	fastfood	188	flower	185
fruit	187	glove	188	hand	182
keys	0	mask	187	mic	0
pen	0	phone	0	scarf	0
sunglasses	0				

***Figure** 5.6 Detector model training on 12 out of 19 different categories chosen by our team.*

5.2.3. Evaluate The Detector and Visualize Predictions

For the evaluation, another file called: metrics.json was also created and saved during the training. A set of model evaluation metrics were defaulted by the Detectron2 library.
We only plotted some of them to show the model performance:

(The metrics are printed out at every iteration of the training loop. The most important ones are the 6 loss values, and below are basic descriptions of them all):

1. *total_loss:* This is a weighted sum of the following individual losses calculated during the iteration. By default, the weights are all one.
2. *loss_cls:* Classification loss in the ROI head. Measures the loss for box classification, i.e., how good the model is at labelling a predicted box with the correct class.
3. *loss_box_reg:* Localization loss in the ROI head. Measures the loss for box localization (predicted location vs true location).
4. *loss_rpn_cls:* Classification loss in the Region Proposal Network. Measures the "objectness" loss, i.e., how good the RPN is at labelling the anchor boxes as foreground or background.

5. *loss_rpn_loc:* Localization loss in the Region Proposal Network. Measures the loss for localization of the predicted regions in the RPN.
6. *loss_mask:* Mask loss in the Mask head. Measures how "correct" the predicted binary masks are.

Evaluate Detector Model

```
import json
import matplotlib.pyplot as plt

experiment_folder = '/content/drive/MyDrive/Colab Notebooks/One Dataset/mask_rcnn_R_50_FPN_3x/2023-03-03-11-51-46'
def load_json_arr(json_path):
    lines = []
    with open(json_path, 'r') as f:
        for line in f:
            lines.append(json.loads(line))
    return lines
experiment_metrics = load_json_arr(experiment_folder + '/metrics.json')
plt.plot(
    [x['iteration'] for x in experiment_metrics],
    [x['total_loss'] for x in experiment_metrics])
plt.plot(
    [x['iteration'] for x in experiment_metrics],
    [x['loss_cls'] for x in experiment_metrics])
plt.plot(
    [x['iteration'] for x in experiment_metrics],
    [x['loss_box_reg'] for x in experiment_metrics])
plt.plot(
    [x['iteration'] for x in experiment_metrics],
    [x['loss_rpn_cls'] for x in experiment_metrics])
plt.plot(
    [x['iteration'] for x in experiment_metrics],
    [x['loss_rpn_loc'] for x in experiment_metrics])
plt.plot(
    [x['iteration'] for x in experiment_metrics],
    [x['loss_mask'] for x in experiment_metrics])

plt.legend(['total_loss', 'loss_cls', 'loss_box_reg', 'loss_rpn_cls', 'loss_rpn_loc', 'loss_mask'], loc='upper left')
plt.show()
```

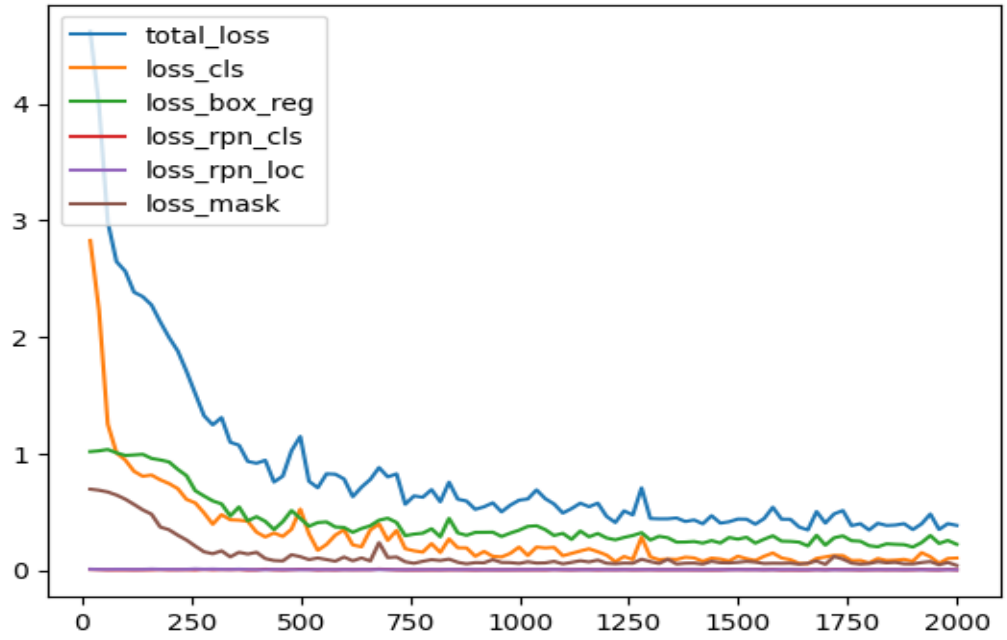


***Figure** 5.7 Plot showing the loss curves of our trained Detector model.*

For an accurate measure of the model detection performance, we also applied the AP (Average Precision) metric on the model and got an AP measure of an ~75, which is considered as good.

We can also evaluate its performance using AP metric implemented in COCO API. Our Detector gives an AP of ~75. Not bad!

```
class Detector:
    def __init__(self):
        self.cfg = get_cfg()
        #loading model config and pretrained model

        self.cfg.merge_from_file(model_zoo.get_config_file(CONFIG_FILE_PATH))
        self.cfg.MODEL.WEIGHTS = "/content/drive/MyDrive/Colab Notebooks/One Dataset/mask_rcnn_R_50_FPN_3x/2023-03-03-11-51-46/mo
        self.cfg.MODEL.ROI_HEADS.NUM_CLASSES = NUM_CLASSES
        self.cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.7
        # Create predictor
        self.predictor = DefaultPredictor(self.cfg)

detector = Detector()
```

```
#Call the COCO Evaluator function and pass the Validation Dataset
evaluator = COCOEvaluator("One Dataset-test", output_dir="./output")
val_loader = build_detection_test_loader(detector.cfg, "One Dataset-test")

#Use the created predicted model in the previous step
print(inference_on_dataset(detector.predictor.model, val_loader, evaluator))
```

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.821
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.898
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.876
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.878
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.878
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.856
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.915
OrderedDict([('bbox', {'AP': 75.3835874035136, 'AP50': 92.24347642186709, 'AP75': 90.87009629098424, 'APs': nan, 'APm': 77.14
618415838713, 'APl': 77.98759115180764, 'AP-book': nan, 'AP-box': 83.70545260046978, 'AP-candy': 72.45751621797181, 'AP-cap':
86.53601680432192, 'AP-cigarette': 51.423866232900664, 'AP-cup': 72.41354178354514, 'AP-envelope': 77.0798760605365, 'AP-fast
food': 80.161938449919, 'AP-flower': 77.76866019701345, 'AP-fruit': 75.09872759135648, 'AP-glove': 66.70410648884702, 'AP-han
d': 79.67041448611559, 'AP-keys': nan, 'AP-mask': 81.5829319291658, 'AP-mic': nan, 'AP-pen': nan, 'AP-phone': nan, 'AP-scar
f': nan, 'AP-sunglasses': nan}), ('segm', {'AP': 85.27585712522134, 'AP50': 93.17102734442358, 'AP75': 92.79751893467827, 'AP
s': nan, 'APm': 82.07158928731704, 'APl': 89.79606531801268, 'AP-book': nan, 'AP-box': 96.99388539807876, 'AP-candy': 80.3002
214507795, 'AP-cap': 99.86347055758208, 'AP-cigarette': 63.45519289238902, 'AP-cup': 88.83370439443816, 'AP-envelope': 90.861
85393618793, 'AP-fastfood': 93.46801305821835, 'AP-flower': 75.99807950831337, 'AP-fruit': 83.10374879330577, 'AP-glove': 73.
80513062159729, 'AP-hand': 81.52675323267326, 'AP-keys': nan, 'AP-mask': 95.10023165909243, 'AP-mic': nan, 'AP-pen': nan, 'AP
-phone': nan, 'AP-scarf': nan, 'AP-sunglasses': nan})])
```

***Figure** 5.8 Evaluating the performance of our trained Detector using the AP (Average Precision) metric implemented in COCO API, the model gave an ~75AP measure.*

To visualize the model detections, we tested our trained model with a 0.7 prediction score threshold on our test dataset, meaning that; only the predictions with a higher than 70% confidence score will be displayed and used.

Test Our Trained Model

```
cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.7
predictor = DefaultPredictor(cfg)
```

```
[03/03 12:07:22 d2.checkpoint.detection_checkpoint]: [DetectionCheckpointer] Loading from One Dataset/mask_rcnn_R_50_FPN_3x/202
3-03-03-11-51-46/model_final.pth ...
```

```
dataset_valid = DatasetCatalog.get(TEST_DATA_SET_NAME)

for d in dataset_valid:
    img = cv2.imread(d["file_name"])
    outputs = predictor(img)

    visualizer = Visualizer(
        img[:, :, ::-1],
        metadata=metadata,
        scale=0.8,
        instance_mode=ColorMode.IMAGE_BW
    )
    out = visualizer.draw_instance_predictions(outputs["instances"].to("cpu"))
    cv2_imshow(out.get_image()[:, :, ::-1])
```

```
[03/03 12:35:21 d2.data.datasets.coco]: Loaded 2186 images in COCO format from /content/drive/MyDrive/One Dataset/test.json
```

***Figure** 5.9 Visualizing some sample outputs from evaluating the trained Detector on our test dataset.*

5.3. THE IMAGE INPAITING MODULE

As stated earlier our aim is to detect the occlusion, remove it an reconstruct the face beneath it for the image inpainting we construct a GAN network the generator consists of a partial and vanilla convolution, two discriminator and a parsing network.

5.3.1. Implementation and Training Details

We loaded our images and resized them to 256x256 px then passed them through the model.

Feature Matching:

Pretrained vgg16 was used, this network will encourage the performance of the generator's outputs to have features that are close enough to the ground truth.

Partial Convolution:

The code starts with the initial methods to build the Pconv layers was defined the methods which initialize the weights, specify, and create the image kernel and the mask kernel padding size and the methods which defines computation that is performed on the input to generate the output of the layer and to calculate the mask ratio which is used to normalize the image output

After all the methods that's related to the custom Partial Convolution were defined.
We built the U-net Architecture which consists of an encoder and decoder.
In this model the encoder stage consists of 8 encoder blocks
The first encoder blocks take the image and the mask along with the filter and the Kernel size specified, the next encoder block will take the outputs of the first encoder block and so on
The decoder takes all the encoded features in the encoder stage and up-sample them to the original image size, the up-sampling is performed on both the original image and mask input, then the up-sampled tensor is concatenated with the output tensor from the corresponding encoder stage.

In that way the partial convolution is performed on the concatenated image and mask tensors which will work on removing the mask gradually.

Vanilla Convolution:

After reading the image and the mask, we concatenate the two and feed them to the vanilla convolution network which is built as a u-net network that consists of an encoder and a decoder.
The encoder consists of 5 encoder blocks (two conv followed by pool layer) and at the fourth and fifth blocks we added a dropout layer.
Dropout regularization is a technique to reduce overfitting, it will randomly drop out some of the neurons during training.
The dropout rate that's used is 0.5

```
conv4 = Conv2D(512, 3, activation='relu', padding='same', kernel_initializer='he_normal')(pool3)
conv4 = Conv2D(512, 3, activation='relu', padding='same', kernel_initializer='he_normal')(conv4)
drop4 = tf.keras.layers.Dropout(0.5)(conv4)
pool4 = MaxPooling2D(pool_size=(2, 2))(drop4)
```

The decoder consists of 4 blocks and the output layer, in this stage we are up sampling. Each block is concatenated with the output of the encoder

```
# Decoding path
up6 = Conv2D(512, 2, activation='relu', padding='same', kernel_initializer='he_normal')(UpSampling2D(size=(2, 2))(drop5))
merge6 = Concatenate()([drop4, up6])
conv6 = Conv2D(512, 3, activation='relu', padding='same', kernel_initializer='he_normal')(merge6)
conv6 = Conv2D(512, 3, activation='relu', padding='same', kernel_initializer='he_normal')(conv6)
```

Throughout the whole network we used ‘relu’ as an activation function. But at the output layer the activation function that’s used is ‘sigmoid’.

The Discriminator:

The discriminator forces the generator to produce an image that is visually plausible.
It consists of convolutional layers and a LeakyRelu activation function. After the convolutional layers the output tensor is flattened and then passed with two dense layers. To reduce the overfitting a dropout was added.

```
discriminator = keras.Sequential(
    [
        keras.Input(shape=(256, 256, 3)),

        layers.Conv2D(32, kernel_size=4, strides=2, padding="same"),
        layers.LeakyReLU(alpha=0.2),

        layers.Conv2D(64, kernel_size=4, strides=2, padding="same"),
        layers.LeakyReLU(alpha=0.2),

        layers.Conv2D(128, kernel_size=4, strides=2, padding="same"),
        layers.LeakyReLU(alpha=0.2),

        layers.Conv2D(256, kernel_size=4, strides=2, padding="same"),
        layers.LeakyReLU(alpha=0.2),

        layers.Conv2D(512, kernel_size=4, strides=2, padding="same"),
        layers.LeakyReLU(alpha=0.2),

        layers.Flatten(),

        layers.Dense(4096 , activation='relu'),
        layers.Dense(4096 , activation='relu'),

        layers.Dropout(0.2),
        layers.Dense(1, activation="sigmoid"),
    ],
    name="discriminator",
)
discriminator.summary()
```

Loss Functions:

We used 6 loss functions which could force the generator to produce a plausible result for the missing parts of the face.

1. **L1** is used to minimize errors, which is a function that calculates the L1 loss between the predicted and true values of a tensor in Keras.

```
def loss_l1(y_true, y_pred):
    """
    Size-averaged L1 loss used in all the losses.
    If size_average is True, the l1 losses are means,
    If size_average is False, the l1 losses are sums divided by norm (should be specified),
        only have effect if y_true.ndim = 4.
    """
    if K.ndim(y_true) == 4:
        # images and vgg features
        return K.mean(K.abs(y_pred - y_true), axis=[1, 2, 3])
    elif K.ndim(y_true) == 3:
        # gram matrices
        return K.mean(K.abs(y_pred - y_true), axis=[1, 2])
    else:
        raise NotImplementedError(
            "Calculating L1 loss on 1D tensors? should not occur for this network"
        )
```

2. **Gram matrix** calculates the gram matrix used in the style loss.

The Gram matrix of a feature map is a matrix that captures the correlations between the different filters in the feature map.

```
def gram_matrix(x, norm_by_channels=False):
    """Calculate gram matrix used in style loss"""
    # Assertions on input
    assert K.ndim(x) == 4, 'Input tensor should be a 4d (B, H, W, C) tensor'
    assert K.image_data_format(
    ) == 'channels_last', "Please use channels-last format"

    # Permute channels and get resulting shape
    x = K.permute_dimensions(x, (0, 3, 1, 2))
    shape = K.shape(x)
    B, C, H, W = shape[0], shape[1], shape[2], shape[3]

    # Reshape x and do batch dot product
    features = K.reshape(x, K.stack([B, C, H * W]))
    gram = K.batch_dot(features, features, axes=2)

    # Normalize with channels, height and width
    gram = gram / K.cast(C * H * W, x.dtype)
    return gram
```

3. **Loss per pixel** computes the pixel wise L1 loss between the predicted image and the ground truth image, only for the pixels outside the hole/mask region.

```
def loss_per_pixel(mask, y_true, y_pred):
    """Pixel L1 loss outside the hole / mask"""
    assert K.ndim(y_true) == 4, 'Input tensor should be 4D (B, H, W, C).'
    return K.mean(K.abs(mask * (y_pred - y_true)), axis=[1, 2, 3])
```

4. **Loss perceptual** this loss function will measure the difference between the predicted image and the ground truth based on their high-level features rather than their pixels values (based on their respective VGG 16 feature representation)

```
def loss_perceptual(vgg_out, vgg_gt, vgg_comp):
    """Perceptual loss based on VGG16, see. eq. 3 in paper"""
    l = 0
    for o, c, g in zip(vgg_out, vgg_comp, vgg_gt):
        l += loss_l1(o, g) + loss_l1(c, g)
    return l
```

5. **Loss style** is used to enforce the similarity between the completed image and the ground truth, in our case this will be computed based on their respective VGG 16 feature representation, the style loss here is computed as the sum of L1 pixel wise loss between the Gram matrix of the ground truth image and the completed image.

```
def loss_style(vgg_out, vgg_gt, vgg_comp):
    """Style loss consisting of two terms: out and comp."""
    style_score = 0.
    for o, g, c in zip(vgg_out, vgg_gt, vgg_comp):
        #print("shapes", o.shape, c.shape, g.shape, "dim", K.ndim(o), K.ndim(c), K.ndim(g))
        gram_gt = gram_matrix(g)
        #print('gram_gt', gram_gt.shape, 'gram_out', gram_matrix(o).shape, gram_matrix(c).shape)
        style_score += loss_l1(gram_matrix(o), gram_gt) + loss_l1(
            gram_matrix(c), gram_gt)
    return style_score
```

6. **Loss tv** computes the total variation loss between the completed image and the inverse binary mask, the main use of total variation loss function is to smooth out the hole regions in the completed image, it's computed as the sum pf L1 pixel wise loss between the image and the spatial gradients.

```
def loss_tv(inv_mask, y_comp):
    """Total variation loss, used for smoothing the hole region"""
    assert K.ndim(y_comp) == 4 and K.ndim(
        inv_mask) == 4, 'Input tensors should be 4D (B, H, W, C).'
    ## Create dilated hole region using a 3x3 kernel of all 1s.
    kernel = tf.ones([3, 3, 3, 3], tf.float32)
    dilated_mask = K.conv2d(inv_mask,
                            kernel,
                            data_format='channels_last',
                            padding='same')

    ## Cast values to be [0., 1.], and compute dilated hole region of y_comp
    dilated_mask = K.cast(K.greater(dilated_mask, 0), 'float32')
    P = dilated_mask * y_comp
    ## Calculate total variation loss
    return loss_l1(P[:, 1:, :, :], P[:, :-1, :, :]) + loss_l1(
        P[:, :, 1:, :], P[:, :, :-1, :])
```

And finally, the total loss was computed for the whole model along the loss that's generated from the two discriminators.

Hyperparameters:

***Table** 5.1 The Hyperparameters we used to train the Vanilla Convolution and of the Partial Convolution model layers.*

	Vanilla	Partial
Number of encoder decoder blocks	6 each encoder (2 conv – 1Pool)	6 encoders and decoders
Activation functions	Relue - Sigmoid	Relue – Leaky relue
Learning rate	0.00005	
Optimizer	Adam	Adam
Batch size	4	4
Training steps (iteration per epoch)	Int(len(trainnfiles)/Batch size)	5000
Dropout rate	0.5	
Regularization		Batch normalization

5.3.2. Separate Training Results

Partial Convolution

We observed that the partial convolution focuses more on the valid regions, it did reconstruct the face, but it missed some details at the affected area specially the edges.

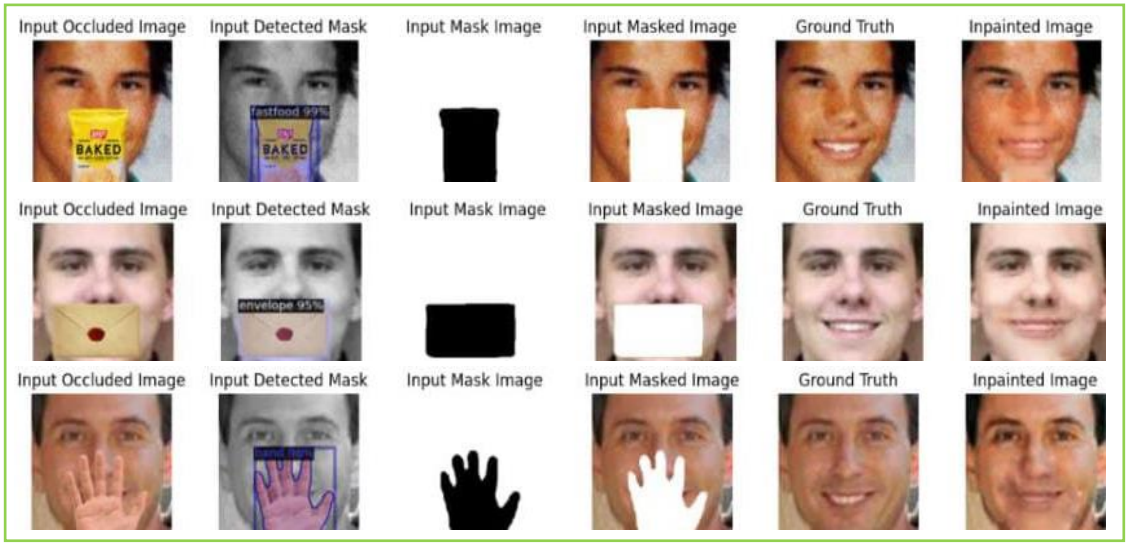


***Figure** 5.10 Inpainting results of the Partial Convolution model.*

Vanilla Convolution

Training with the two discriminators the model predicted the images well, but the blurriness was still notable, and the edges of the face weren't constructed well.

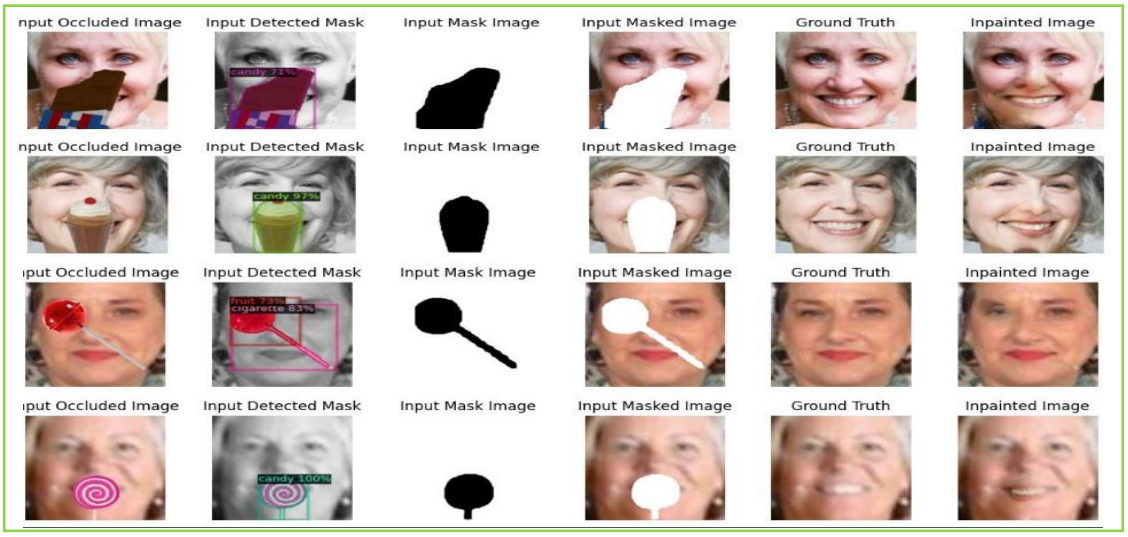


***Figure** 5.11 Inpainting results of the Vanilla Convolution model.*

Chapter 6

The Model Results

6.1. OUR MODEL'S RESULTS

6.1.1. Experiments

We concatenate the output of the partial convolution with the output of the vanilla convolution and the result of this concatenation is fed into an atrous convolution block, then the output of the atrous is used as input for the decoder.
An important step was to make sure that all the networks had the same shape of output so we could avoid a mismatch.
We trained the model with all loss functions and discriminators that had been stated earlier.

6.1.2. Model Results

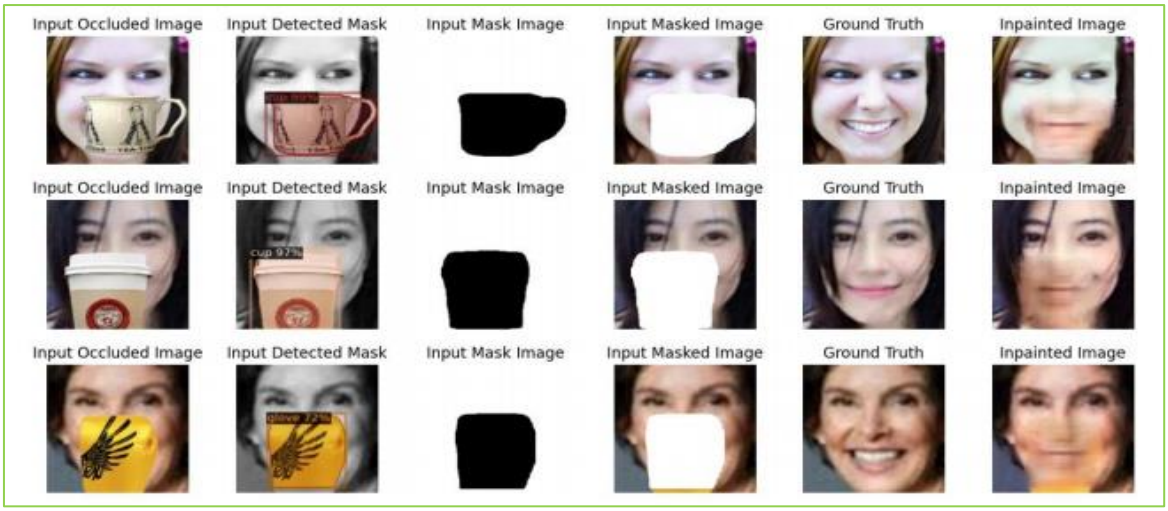


***Figure** 5.12 Inpainting results of our model.*

As we can see the model's performance was higher on the smaller occlusions but as we increase the training the model was enhancing the result's it generates.

6.2. FUTURE WORK

6.2.1. Face Parsing Network

We found that the global D is not effective in ensuring the consistency of fine details in the generated image.

The main reason that a parsing network was introduced is that a global discriminator is not enough to ensure the consistency of the image considering the details it could give a realistic completion but not identical or close to the unmasked one.

- It is used for regularization
- Segment labels for every main component of the face
- It predicts the label for each pixel
- In the paper it achieved f score of 0.851
- Improve the quality of face completion and can be improved with hyperparameter tuning

After training, the parsing network remains fixed in the generation network framework.

We compare the result of the unmasked face the ground truth with the parsing on the generated faces, The parsing will back-propagate to the generator to regularize face completion [1].

6.2.2. Attention Refine Network

The attention model was introduced to borrow and copy all the feature information from background patches which allows us to generate masked patched also allows us to keep the image coherent [4].

6.3. CONCLUSION

In this paper we documented the project of detecting and reconstructing the face under mask and occlusions. We show that our model using the vanilla and the partial convolution along with the discriminators gave a good and a promising result, which could be enhanced and improved. Our model acts well when inpainting small, impaired regions, but acts poorly when these regions get bigger, it couldn't inpaint large holes efficiently. However, our model was capable of successfully detecting and removing 19 different occlusions and reconstructing the face image as long as the occlusion covers a reasonable region of the face .In conclusion our model outperforms many approaches by detecting more than 19 objects which is not a small number of occlusions.

References

[1] S. L. J. Y. M.-H. Y. Yijun Li, "Generative Face Completion," 19 Apr 2017. [Online]. Available: .https://arxiv.org/pdf/1704.05838.pdf

[2] Morol, "A Deep Learning Framework to Reconstruct .Miraj and . M. K .Das, M. A. I .Modak, S. S .G .Face under Mask," 23 Mar 2022. [Online]. Available: https://arxiv.org/abs/2203.12482

[3] JAVED, S. BAE and J. YI, "Effective Removal of User-Selected Foreground," 24 Jun .N. U. DIN, K .arnumber=9114981&=2020. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp

[4] L. FEIHONG, C. HANG, L. KANG, D. QILIANG, Z. JIAN, Z. KAIPENG and H. HONG, "Toward High- [Online]. Available: ",quality Face-Mask Occluded Restoration .https://dl.acm.org/doi/pdf/10.1145/3524137

[5] Huang, G. Jia, Z. Chai and X. Wei, "Contrastive Attention Network with Dense .X. Ma, X. Zhou, H Field Estimation for Face Completion," 20 Dec 2021. [Online]. Available: .https://arxiv.org/pdf/2112.10310.pdf

[6] T. Guo, Y. Kim, H. Zhang, D. Qian, B. Yoo, J. Xu, D. Zou, J. J. Han and C. Choi, "Residual Encoder .Decoder Network," 2018. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/12268

[7] H. Yoshihashi, N. Ienaga and M. Sugimoto, "GAN-based Face Mask Removal using Facial Landmarks and Pixel," 2022. [Online]. Available: .https://drive.google.com/file/d/1HSeAjHmeIHMVd69fQRyGLgU7xv8cEMkd/view?usp=drivesdk

[8] Kannala, "MASK-RCNN AND U-NET ENSEMBLED FOR NUCLEI .A. O. Vuola, S. U. Akram and J .SEGMENTATION," 29 Jan 2019. [Online]. Available: https://arxiv.org/pdf/1901.10170.pdf

[9] Catanzaro, "Image Inpainting for Irregular .G. Liu, F. A. Reda, K. J. Shih, T. C. Wang, A. Tao and B .Holes Using," 15 Dec 2018. [Online]. Available: https://arxiv.org/pdf/1804.07723.pdf

[10] Goodfellow, J. P. -Abadie, M. Mirza, S. Ozair, A. Courville, Y. Bengio, D. W. Farley and B. Xu, .I. J ."Generative Adversarial Nets," 10 Jun 2014. [Online]. Available: https://arxiv.org/pdf/1406.2661.pdf

[11] .Wang, L. Wang, G. Wang, J .Ting, X .Ma, A. Shahroudy, B. Shua, L .Kuen, L .J. Gu, Z. Wang, J Ca and T. Chen, "Recent Advances in Convolutional Neural Networks," 19 Oct 2019. [Online]. .Available: https://arxiv.org/pdf/1512.07108.pdf%C3%A3%E2%82%AC%E2%80%9A

[12] Ronneberger, O., Fischer, P. and Brox, T. (2015) *U-Net: Convolutional Networks for Biomedical Image Segmentation*, *arXiv.org*. Available at: https://arxiv.org/abs/1505.04597

[13] *Karras, T. et al. (2018) Progressive growing of GANs for improved quality, stability, and Variation, arXiv.org. Available at: https://arxiv.org/abs/1710.10196*

[14] Zhang, Z., Song, Y. and Qi, H. (2017) *Age progression/regression by conditional adversarial autoencoder*, *arXiv.org*. Available at: https://arxiv.org/abs/1702.08423

[15] Kipr (2010) *Opencv/haarcascade_frontalface_default.xml at master · KIPR/opencv*, *GitHub*. Available at: https://github.com/kipr/opencv/blob/master/data/haarcascades/haarcascade_frontalface_default.xml

[16] *Releases* (2020) *OpenCV*. Available at: https://opencv.org/releases/

[17] *Handbook* (no date) *Pillow (PIL Fork)*. Available at: https://pillow.readthedocs.io/en/stable/handbook/index.html

[18] Yuxin Wu *et al.* (2019) *Facebookresearch/Detectron2: Detectron2 is a platform for object detection, segmentation and other visual recognition tasks.*, *GitHub*. Available at: https://github.com/facebookresearch/detectron2

[19] J.L. *et al.* (2018) *Create your own coco-style datasets*, *waspinator*. Available at: https://patrickwasp.com/create-your-own-coco-style-dataset/